



Reservoir of diverse adaptive learners and stacking fast hoeffding drift detection methods for evolving data streams

Ali Pesaranghader¹  · Herna Viktor¹  ·
Eric Paquet^{1,2}

Received: 30 March 2017 / Accepted: 18 May 2018 / Published online: 1 June 2018
© The Author(s) 2018

Abstract The last decade has seen a surge of interest in adaptive learning algorithms for data stream classification, with applications ranging from predicting ozone level peaks, learning stock market indicators, to detecting computer security violations. In addition, a number of methods have been developed to detect concept drifts in these streams. Consider a scenario where we have a number of classifiers with diverse learning styles and different drift detectors. Intuitively, the current ‘best’ (classifier, detector) pair is application dependent and may change as a result of the stream evolution. Our research builds on this observation. We introduce the TORNADO framework that implements a reservoir of diverse classifiers, together with a variety of drift detection algorithms. In our framework, all (classifier, detector) pairs proceed, in parallel, to construct models against the evolving data streams. At any point in time, we select the pair which currently yields the best performance. To this end, we introduce the CAR measure, which is employed to balance classification, adaptation and resource utilization requirements. We further incorporate two novel stacking-based drift detection methods, namely the FHDDMS and FHDDMS_{add} approaches. The experimental evaluation confirms that the current ‘best’ (classifier, detector) pair is not only heavily dependent on the characteristics of the stream, but also that this selection evolves as the stream flows.

Editors: Toon Calders and Michelangelo Ceci.

Electronic supplementary material The online version of this article (<https://doi.org/10.1007/s10994-018-5719-z>) contains supplementary material, which is available to authorized users.

✉ Ali Pesaranghader
apesaran@uottawa.ca

Herna Viktor
hviktor@uottawa.ca

Eric Paquet
eric.paquet@nrc-cnrc.gc.ca

¹ School of Electrical Engineering and Computer Science, University of Ottawa, Ottawa, ON K1N 6N5, Canada

² National Research Council (NRC) of Canada, 1200 Montreal Road, Ottawa, ON K1A 0R6, Canada

Further, our FHDDMS variants detect concept drifts accurately in a timely fashion while outperforming the state-of-the-art.

Keywords Data stream mining · Online learning · Adaptive learning · Concept drift · Drift detection · Classification · Hoeffding’s inequality

1 Introduction

The last decade has seen a rapid increase in the amount of massive, rapidly evolving data streams. Today’s decision makers require new solutions to comprehend these fast-evolving knowledge sources in near real-time. That is, they need near-instant models to aid them to detect traffic congestion, to analyze smart phone usage patterns, to track the trends in the online sales of merchandise, for mobile crowd sensing, or to trace the spread of ideas, opinions and movements in social networks. This fact has resulted in a surge of interest in adaptive learning algorithms for data stream classification, that are able to learn incrementally and to rapidly adapt to changes in the data (so-called concept drift) (Žliobaite et al. 2016). Such approaches take into consideration that the learning environment is non-stationary and, as a result, build models that evolve over time, as the data arrive.

A number of incremental learners, such as the Hoeffding Tree (HT)¹ (Domingos and Hulten 2000), Naive Bayes, Perceptron (Bifet et al. 2010; Freund and Schapire 1999) and K-Nearest Neighbors (K-NN) methods have been developed, in order to learn from data streams. Intuitively, the performance of an individual classifier may vary as a stream evolves. Also, the difference in learning styles may cause a specific classifier to excel against one stream, while failing to accurately model another. Further, no single concept drift detection technique outperforms others in all settings. Rather, the current ‘best’ pair of classifier and drift detector also changes, as the stream evolves.

Based on these observations, we introduce the TORNADO framework. In our framework a reservoir of classifiers with diverse learning styles co-exists, together with a number of different drift detector algorithms. These classifiers learn incrementally, in parallel. The TORNADO framework operates as follows. Each of the classifiers in the reservoir incrementally accepts the incoming instances, one at a time, and proceeds to build a model. Each classifier is combined with each one of the drift detectors. That is, classifier C_1 is combined with drift detectors D_1, D_2, \dots, D_m , to form pairs $(C_1, D_1), (C_1, D_2), \dots, (C_1, D_m)$, classifier C_2 is combined with drift detectors D_1, D_2, \dots, D_m , to form pairs $(C_2, D_1), (C_2, D_2), \dots, (C_2, D_m)$, and so on. Our CAR measure, as introduced in Sect. 3, is used in order to rank the current best performing (classifier, detector) pair. The CAR measure not only considers the classification error-rate, but also takes into consideration the memory usage, runtime as well as the drift detection delay, together with the number of false positives and false negatives.

In addition, we also incorporate two new drift detection methods into the TORNADO framework. They extend the FHDDM algorithm, as introduced in (Pesaranghader and Viktor 2016), in two ways. Firstly, we introduce the FHDDMS algorithm that creates a so-called “stack” of sliding windows of different sizes. The windows monitor the streams using bitmaps and alarm for concept drift using threshold values. The intuition behind this approach is that, by utilizing the windows of various sizes to monitor the stream, concept drift is detected faster and more accurately. In addition, we present FHDDMS_{add}, a variant of FHDDMS, that employs data summaries, instead of bitwise operations.

¹ The algorithm is also known as Very Fast Decision Tree (VFDT) in the literature.

Our experimental evaluation against synthetic and real-world data streams confirms that the current best (classifier, drift detector) pair evolves as the characteristics of the stream changes. In addition, our FHDDMS methods detect changes faster and more accurate, with shorter delays, fewer false positives and false negatives, when compared to the state-of-the-art.

This paper is organized as follows. Section 2 discusses related work. In Sect. 3, we introduce our CAR measure. This is followed, in Sect. 4, by an overview of the TORNADO framework. Section 5 presents the FHDDMS and FHDDMS_{add} algorithms. In Sect. 6, we detail our experimental setup and results. Section 7 provides a detailed discussion regarding our experiments. Section 8 concludes the paper and highlights future work.

2 Related work

This section discusses related work on performance measures for data stream mining, by focusing on classification and adaptation measures in a streaming setting, which we use as a foundation for defining the CAR measure in Sect. 3. In addition, we review the state-of-the-art in terms of drift detection algorithms.

2.1 Performance measures for adaptive online learning

Researchers agree that the evaluation of data stream algorithms is a complex task. This fact is due to many challenges, including the presence of concept drift, limited processing time in real-world applications and the need for time-oriented evaluation, amongst others (Gama et al. 2004). The error-rate (or accuracy) is most often used as the defining measure of the classification performance for evaluating learning algorithms in most streaming studies (Hulten et al. 2001; Gama et al. 2004, 2006; Bifet and Gavalda 2007; Huang et al. 2015; Baena-Garcia et al 2006). The error-rate is calculated incrementally using either the prequential or hold-out evaluation procedures (Bifet and Kirkby 2009). The interplay between the error-rate and other factors, such as memory usage and runtime considerations, has received limited attention. Bifet et al. (2009) considered the memory, time and accuracy measures separately, in order to compare the performances of ensembles of classifiers. Bifet et al. (2010) further introduced the RAM-Hour measure, where every RAM-Hour equals to 1 GB of RAM occupied for one hour, to compare the performances of three versions of perceptron-based Hoeffding Trees. Pesaranghader et al. (2016) introduced the EMR measure which combines error-rate, memory usage and runtime for evaluating and ranking learning algorithms.

Žliobaite et al. (2015a) introduced the return on investment (ROI) measure to determine whether the *adaptation* of a learning algorithm is beneficial. They concluded that adaptation should only take place if the expected gain in performance, measured by accuracy, exceeds the cost of other resources (e.g. memory and time) required for adaptation. In their work, the ROI measure was used to indicate whether an adaptation to a concept drift is beneficial, over time. Olorunnimbe et al. (2015) extended the above-mentioned ROI measure, in order to dynamically adapt the number of base learners in online bagging ensembles. Pesaranghader and Viktor (2016) proposed an approach to count true positive (TP), false positive (FP), and false negative (FN) of drift detection, in order to evaluate the performances of concept drift detectors. They introduced the acceptable delay length notion as a threshold that determines how far a detected drift could be from the real location of drift to be considered as a true positive.

As explained above, the performance measures of classification and adaptation have been often used, separately, to evaluate adaptive learning algorithms against evolving data streams. To date, no single measure that considers classification, adaptation, and resource consumption together, has been developed. Such a measure would allow one to assess the “big picture”, in terms of the costs and benefits of a specific learning and adaptation strategy. In Sect. 3, we introduce the CAR measure in order to address this deficiency.

2.2 Drift detection methods

Gama et al. (2014) categorized concept drift detectors into three general groups, as follows:

1. *Sequential Analysis based Methods* sequentially evaluate prediction results as they become available, and alarm for drifts when a pre-defined threshold is met. The Cumulative Sum (CUSUM) and its variant Page-Hinkley (PH) (Page 1954), as well as Geometric Moving Average (GMA) (Roberts 2000) are members of this group.
2. *Statistical based Approaches* probe the statistical parameters such as mean and standard deviation of prediction results to detect drifts in a stream. The Drift Detection Method (DDM) (Gama et al. 2004), Early Drift Detection Method (EDDM) (Baena-García et al 2006) and Exponentially Weighted Moving Average (EWMA) (Ross et al. 2012) are members of this group.
3. *Windows based Methods* usually use a fixed reference window summarizing the past information and a sliding window summarizing the most recent information. A significant difference between the distributions of these two windows suggests the occurrence of a drift. Statistical tests or mathematical inequalities, with the null-hypothesis indicating that the distributions are equal, are thus employed. Kifer’s (Kifer et al. 2004), Nishida’s (Nishida and Yamauchi 2007), Bach’s (Bach and Maloof 2008), the Adaptive Windowing (ADWIN) (Bifet and Gavalda 2007), SeqDrift detectors (Sakthithasan et al. 2013; Pears et al. 2014), Drift Detection Methods based on Hoeffding’s Bound (HDDM_{A-test} and HDDM_{W-test}) (Frías-Blanco et al. 2015), and Adaptive Cumulative Windows Model (ACWM) (Sebastião et al. 2017) are members of this family.

CUSUM and its variant Page-Hinkley (PH) are some of the pioneer methods in the community. DDM, EDDM, and ADWIN have frequently been considered as benchmarks in the literature (Huang et al. 2015; Frías-Blanco et al. 2015; Baena-García et al 2006; Nishida and Yamauchi 2007; Bifet and Gavalda 2007; Pesaranghader and Viktor 2016). SeqDrift2 and HDDMs are recently proposed methods, and have shown comparable results to the other benchmarks. We, therefore, consider all these methods for our experimental evaluation, and we briefly describe them as follows.

CUSUM: Cumulative Sum—CUSUM, by Page (1954), is a sequential analysis technique that alarms for a change when the mean of the input data significantly deviates from zero. The input of CUSUM can be any filter residual; for instance, the prediction error from a Kalman filter (Gama et al. 2014). The CUSUM test is in the form of $g_t = \max(0, g_{t-1} + (x_t - \delta))$, and it alarms for a concept drift when $g_t > \lambda$. In this test, x_t is the currently observed value, δ specifies the magnitude of changes that are allowed, while $g_0 = 0$ and λ is a user-defined threshold. The accuracy of CUSUM depends on the values of parameters δ and λ . Lower values of δ result in faster detection, at the cost of an increased number of false alarms.

PH: Page-Hinkley—PH, by Page (1954), is a variant of CUSUM typically used for change detection in signal processing applications (Gama et al. 2014). The test variable m_T is defined as a cumulative difference between the observed values and their mean until the

current time T ; and calculated by $m_T = \sum_{t=1}^T (x_t - \bar{x}_T - \delta)$, where $\bar{x} = \frac{1}{T} \sum_{t=1}^T x_t$ and δ defines the allowed magnitude of changes. The PH method also updates the minimum m_T , denoted as M_T , using $M_T = \min(m_t, t = 1 \dots T)$. A significant difference between m_T and M_T , i.e. $PH_T : m_T - M_T > \lambda$ where λ is a user-defined threshold, implies a concept drift. A large value of λ typically causes fewer false alarms, but it may increase false negative rate.

DDM: Drift Detection Method—DDM, by Gama et al. (2004), monitors the error-rate of the classification model to detect drifts. On the basis of PAC learning model (Mitchell 1997), the method considers that the error-rate of a classifier decreases or stays constant as the number of instances increases. Otherwise, it suggests the occurrence of a drift. Consider p_t as the error-rate of the classifier with a standard deviation of $s_t = \sqrt{(p_t(1 - p_t)/t)}$ at time t . As instances are processed, DDM updates two variables p_{min} and s_{min} when $p_t + s_t < p_{min} + s_{min}$. DDM warns for a drift when $p_t + s_t \geq p_{min} + 2 * s_{min}$, and it detects a drift when $p_t + s_t \geq p_{min} + 3 * s_{min}$. The p_{min} and s_{min} are reset when a drift is detected.

EDDM: Early Drift Detection Method—EDDM, by Baena-Garcia et al (2006), evaluates the distances between wrong predictions to detect concept drifts. The algorithm is based on the observation that a drift is more likely to occur when the distances between errors are smaller. EDDM calculates the average distance between two recent errors, i.e. p'_t , with its standard deviation s'_t at time t . It updates two variables p'_{max} and s'_{max} when $p'_t + 2 * s'_t > p'_{max} + 2 * s'_{max}$. The method warns for a drift when $(p'_t + 2 * s'_t) / (p'_{max} + 2 * s'_{max}) < \alpha$, and indicates that a drift occurred when $(p'_t + 2 * s'_t) / (p'_{max} + 2 * s'_{max}) < \beta$. The authors set α and β to 0.95 and 0.90, respectively. The p'_{max} and s'_{max} are reset only a drift is detected.

HDDMs—HDDM_{A-test} and HDDM_{W-test} are proposed by Frías-Blanco et al. (2015). The former compares the moving averages to detect drifts. The latter uses the EMWA forgetting scheme (Ross et al. 2012) to weight the moving averages. Then, weighted moving averages are compared to detect concept drifts. For both cases, the Hoeffding inequality (Hoeffding 1963) is used to set an upper bound to the level of difference between averages. The authors noted that the first and the second methods are ideal for detecting abrupt and gradual drifts, respectively.

ADWIN: Adaptive Sliding Window—ADWIN, by Bifet Bifet and Gavaldà (2007), slides a window w as the predictions become available, in order to detect drifts. The method examines two sub-windows of sufficient width, i.e. w_0 with size n_0 and w_1 with size n_1 , of w , where $w_0 \cdot w_1 = w$. A significant difference between the means of two sub-windows indicates a concept drift, i.e. $|\hat{\mu}_{w_0} - \hat{\mu}_{w_1}| \geq \varepsilon$ where $\varepsilon = \sqrt{\frac{1}{2m} \ln \frac{4}{\delta'}}$, m is the harmonic mean of n_0 and n_1 , $\delta' = \delta/n$. Here δ is the confidence level while n is the size of window w . After a drift is detected, elements are removed from the tail of the window until no significant difference is seen.

SeqDrift2—SeqDrift2, by Pears et al. (2014), uses the reservoir sampling method (Vitter 1985), as an adaptive sampling strategy, for random sampling from input data. SeqDrift2 stores entries into two repositories called *left* and *right*. As entries are processed over time, the left repository contains a combination of older and new entries by applying the reservoir sampling strategy. The right repository collects the new arriving entries. SeqDrift2 subsequently finds an upper bound for the difference in between the means of the two repositories, i.e. $\hat{\mu}_l$ for the left repository and $\hat{\mu}_r$ for the right repository, using the Bernstein inequality (Bernstein 1946). Finally, a significant difference between the two means suggests a concept drift.

Discussion—CUSUM and Page-Hinkley (PH) detect concept drift by calculating the difference of observed values from the mean and alarm for a drift when this value is larger than a user-defined threshold. These algorithms are sensitive to the parameter values, resulting a tradeoff between false alarms and detecting true drifts. Recall that DDM, EDDM and HDDM maintain sets of variables, in order to monitor a stream for concept drift. The ADWIN and SeqDrift2 methods, on the other hand, maintain more than one subset of the stream, either using windowing or repositories. DDM and EDDM have lower memory footprints as they only maintain a small number of variables (Gama et al. 2014). These two approaches also require less execution runtime to update the values of the variables for drift detection. However, EDDM may frequently alarm for concept drift during the early stages of learning, since the distance between wrong predictions is small. HDDM employs the Hoeffding inequality in order to detect concept drift. ADWIN and SeqDrift2 generally require more memory for storing prediction results, as maintained within sliding windows or repositories. They are also computationally more expensive, due to the sub-window compression or reservoir sampling procedures. Recall that the SeqDrift2 algorithm of Pears et al. (2014) employs the Bernstein inequality in order to detect concept drift. SeqDrift2 uses the sample variance, and assumes that the sampled data follow a normal distribution. It follows that this assumption may be too restrictive, in real-world domains. Further, the Bernstein inequality is conservative and requires a variance parameter, in contrast to, for instance, the Hoeffding inequality. These shortcomings may lead to longer detection delays and a potential loss of accuracy. In summary, our preliminary experimentation confirmed that the aforementioned methods may cause long detection delay, high false positive and high false negative rates. In Sect. 5, we will introduce our new Stacking Fast Hoeffding Drift Detection Method (FHDDMS), that extends our earlier introduced Fast Hoeffding Drift Detection Method (FHDDM) technique (Pesaranhader and Viktor 2016). FHDDM slides a window over the stream, in order to detect concept drift. We maintain two variables, namely the mean of elements inside the window at the current time and the maximum mean observed so far. FHDDM subsequently employs the Hoeffding inequality to detect drifts. Our approach thus differs from HDDM, in that we use a sliding window and only maintain two variables.

3 The CAR performance measure

This section presents our CAR measure, which is employed in order to balance classification, adaptation and resource utilization requirements. The motivation for introducing this measure is as follows. Intuitively, as illustrated in Fig. 1, a change in the data distribution, as caused by a concept drift, may result in an increase or decrease of the error-rates for different types of classifiers (Olorunnimbe et al. 2015). We simulated a number of drift points and showed that, as a concept drift occurs, the classifier with the lowest error-rate changes. Consequently, it follows that a learning system where different types of classifiers co-exist and where the model, from the current “best” learner is provided to the users, may hold much value.

However, following an “*error-rate-only*” approach is not beneficial in all settings. For instance, in an emergency response scenario, the response time, i.e. the time required to present a model to the users, may be the most important criterion. That is, users may be willing to sacrifice accuracy for speed and partial information. Further, consider the area of pocket (or mobile) data mining, which has much application in areas such as defense and environmental impact assessment (Gaber et al. 2014). Here, the memory resources may be limited, due to connection issues, and thus reducing the memory footprint is also of importance (Olorunnimbe et al. 2015).

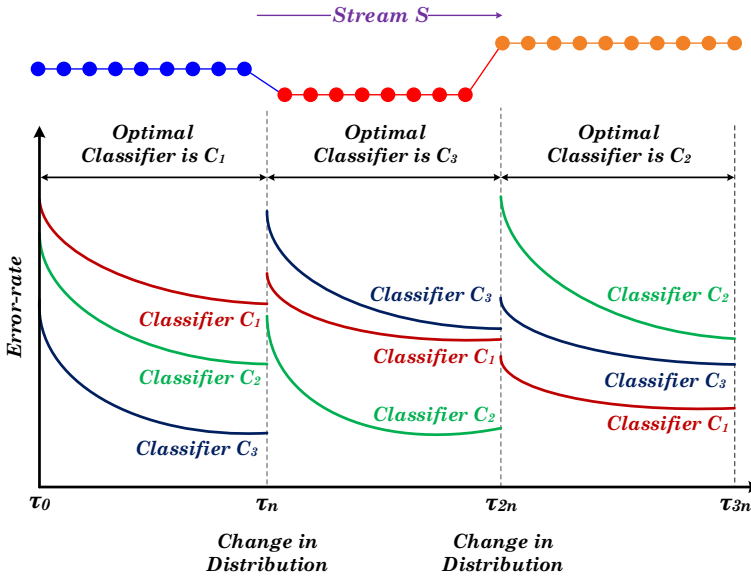


Fig. 1 Illustration of error-rate and distributional change interplay

To address this challenge, the EMR measure was proposed for evaluating the overall performance of learning algorithms based on Error-rate, Memory usage, and Runtime (Pesaranghader et al. 2016). In this approach, classifiers receive their scores based on their current EMR values which are used to rank them. The EMR measure does not fully reflect the performance of learning methods in an environment with concept drift. For instance, consider medical applications, where an adaptive learning algorithm must handle concept drifts very rapidly. A scenario where an airplane is on autopilot is another application where adaptive algorithms, that frequently alarm falsely for concept drifts, are not considered suitable. In such contexts, the EMR measure does not effectively represent the overall performance of adaptive learning algorithms because the detection delay as well as the number of false alarms are both ignored. Subsequently, it is beneficial to represent the performances of the learners, their adaptability as well as their resource usage statistics, by a single measure.

The CAR Measure—We introduce the CAR measure which not only considers the classification error-rates, memory usages, and runtimes but also the drift detection delays, false positives and false negatives. The CAR measure, as defined in Eq. (3.1), consists of three components namely *Classification*, *Adaptation*, and *Resource Consumption*. The classification part consists of the error-rate (E_C) of classifier C , the adaptation part represents the detection delay (D_D), false positive (FP_D) and false negative (FN_D) of drift detector D , while the resource consumption part is associated with the memory consumption ($M_{(C,D)}$) and runtime ($R_{(C,D)}$) of the (classifier, detector) pair. Please note that, in Eq. (3.1), the \diamond and \oplus symbols only represent the combination of three components.

$$\begin{aligned}
 \text{CAR}_{(C,D)} &:= \text{Classification}_C \diamond \text{Adaptation}_D \diamond \text{Resource Use}_{(C,D)} \\
 &:= E_C \diamond (D_D \oplus FP_D \oplus FN_D) \diamond (M_{(C,D)} \oplus R_{(C,D)}) \tag{3.1}
 \end{aligned}$$

The score associated with the pair (C, D) is obtained from Eq. (3.2). The equation implies that a pair with a high CAR has a low score, i.e.

$$\text{Score}_{(C,D)} := 1 - \text{CAR}_{(C,D)} \tag{3.2}$$

In order to compute the CAR measure, a matrix containing the classification, adaptation, and resource consumption metrics of all (classifier, detector) pairs is created, as the first instance arrives. Note that the current value for the CAR measure is continuously re-calculated during online learning. That is, it is reset *every time* a new instance is processed. There are n classifiers and m drift detectors, which means that $n \times m$ pairs, are considered concurrently. The measures associated with each pair for the t th instance are placed, row-by-row, into a matrix M^t , as shown in Eq. (3.3). This matrix is defined as follows:

$$M^t = \begin{bmatrix} E_{C_1}^t & D_{D_1}^t & FP_{D_1}^t & FN_{D_1}^t & (M_{C_1}^t + M_{D_1}^t) & (R_{C_1}^t + R_{D_1}^t) \\ E_{C_1}^t & D_{D_2}^t & FP_{D_2}^t & FN_{D_2}^t & (M_{C_1}^t + M_{D_2}^t) & (R_{C_1}^t + R_{D_2}^t) \\ E_{C_1}^t & D_{D_3}^t & FP_{D_3}^t & FN_{D_3}^t & (M_{C_1}^t + M_{D_3}^t) & (R_{C_1}^t + R_{D_3}^t) \\ \dots & \dots & \dots & \dots & \dots & \dots \\ E_{C_n}^t & D_{D_m}^t & FP_{D_m}^t & FN_{D_m}^t & (M_{C_n}^t + M_{D_m}^t) & (R_{C_n}^t + R_{D_m}^t) \end{bmatrix} \tag{3.3}$$

Subsequently, the elements of the matrix are normalized, column-by-column, using the ‘min-max’ scaling approach. The resulting normalized matrix, \overline{M}^t , is defined in Eq. (3.4):

$$\overline{M}^t = \begin{bmatrix} \frac{E_{C_1}^t}{E_{C_1}^t} & \frac{D_{D_1}^t}{D_{D_1}^t} & \frac{FP_{D_1}^t}{FP_{D_1}^t} & \frac{FN_{D_1}^t}{FN_{D_1}^t} & \frac{(M_{C_1}^t + M_{D_1}^t)}{(M_{C_1}^t + M_{D_1}^t)} & \frac{(R_{C_1}^t + R_{D_1}^t)}{(R_{C_1}^t + R_{D_1}^t)} \\ \frac{E_{C_1}^t}{E_{C_1}^t} & \frac{D_{D_2}^t}{D_{D_2}^t} & \frac{FP_{D_2}^t}{FP_{D_2}^t} & \frac{FN_{D_2}^t}{FN_{D_2}^t} & \frac{(M_{C_1}^t + M_{D_2}^t)}{(M_{C_1}^t + M_{D_2}^t)} & \frac{(R_{C_1}^t + R_{D_2}^t)}{(R_{C_1}^t + R_{D_2}^t)} \\ \frac{E_{C_1}^t}{E_{C_1}^t} & \frac{D_{D_3}^t}{D_{D_3}^t} & \frac{FP_{D_3}^t}{FP_{D_3}^t} & \frac{FN_{D_3}^t}{FN_{D_3}^t} & \frac{(M_{C_1}^t + M_{D_3}^t)}{(M_{C_1}^t + M_{D_3}^t)} & \frac{(R_{C_1}^t + R_{D_3}^t)}{(R_{C_1}^t + R_{D_3}^t)} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ \frac{E_{C_n}^t}{E_{C_n}^t} & \frac{D_{D_n}^t}{D_{D_n}^t} & \frac{FP_{D_n}^t}{FP_{D_n}^t} & \frac{FN_{D_n}^t}{FN_{D_n}^t} & \frac{(M_{C_n}^t + M_{D_n}^t)}{(M_{C_n}^t + M_{D_n}^t)} & \frac{(R_{C_n}^t + R_{D_n}^t)}{(R_{C_n}^t + R_{D_n}^t)} \end{bmatrix} \tag{3.4}$$

A weight is assigned to each measure. The weights are combined into a weight vector \vec{w} which is defined in Eq. (3.5). The elements of the weight vector \vec{w} are the weights associated with the classification error-rate, the detection delay, the false positive rate, the false negative rate, the memory usage, and the runtime. Each weight emphasizes the importance of a particular measure in the evaluation process.

Note that, since each column of the CAR measure is normalized, as illustrated by Eq. (3.4), the corresponding components, such as the memory consumption and the runtime, are also normalized. This implies that the two measures are scale invariant and remain unaffected by change of units, e.g. a memory unit change from kilobytes to megabytes. In our experimental evaluation, we not only show the CAR scores, as a holistic measure, but also include the error rates, memory usages and runtimes. These measures are explicitly shown, to enable the reader to fully evaluate the performance of (classifier, detector) pairs.

$$\vec{w} = [w_e \ w_d \ w_{fp} \ w_{fn} \ w_m \ w_r]^T \tag{3.5}$$

The CAR measures and scores for the $n \times m$ pairs are evaluated with Eqs. (3.6) and (3.7), respectively. Please note that $J_{n \times m, 1}$ is a vector that solely consists of unit entries.

$$\text{CAR}_{n \times m, 1}^t = \frac{\overline{M^t} \cdot \vec{w}}{J_{1,6} \cdot \vec{w}} \quad (3.6)$$

$$\text{Score}_{n \times m, 1}^t = J_{n \times m, 1} - \text{CAR}_{n \times m, 1}^t \quad (3.7)$$

The index of the classifier, that is recommended at time t , $\text{Index}_{\text{opt}, t}$, is defined by Eq. (3.8)²:

$$\text{Index}_{\text{opt}} = \text{imax}(\text{Score}_{n \times m, 1}^t) \quad (3.8)$$

In summary, the CAR measure and scores are calculated as follow:

1. Initialize the matrix M , and the weight vector \vec{w} ,
2. Replace the elements of matrix M row-by-row (i.e. for every pair) as the next instance arrives,
3. Normalize the elements of matrix M column-by-column using ‘min-max’ scaling,
4. Calculate the CAR measures and the scores of pairs with Eqs. (3.6) and (3.7),
5. Rank the pairs based on their scores,
6. Recommend the pair with the highest score to the user,
7. Repeat steps 2 to 6.

One should notice that the weights are application dependent. For instance, if the memory resources are limited, such as in the case of a pocket data mining scenario (Gaber et al. 2014), the value of w_m should be set to a higher value. On the other hand, if memory is abundant, but accuracy and speed of model construction are important, the w_m value may be decreased (or even set to zero). In medical applications, where reacting rapidly to concept drifts is critical, w_d may be the dominant weight.

4 TORNADO: A reservoir of diverse learning strategies

In this section, we introduce the TORNADO framework which is outlined in Fig. 2. We implemented our framework using the Python programming language. Recall that, in our framework, a number of distinct pairs of classifiers and drift detectors are executed in *parallel*, against the same data stream. In this figure, C_n and D_m represent the n th classifier and the m th detector, respectively. A number of classifiers with different learning styles are implemented. Currently, the Naive Bayes (NB), Decision Stump (DS), Hoeffding Tree (HT) (Domingos and Hulten 2000), Perceptron (PR) (Bifet et al. 2010), and K-Nearest Neighbors (K-NN) learning algorithms are available. Furthermore, various concept drift detection methods are provided. Specifically, we have implemented Cumulative Sum (CUSUM) and its variant Page-Hinkley (PH) (Page 1954), Drift Detection Method (DDM) (Gama et al. 2004), Early Drift Detection Method (EDDM) (Baena-Garcia et al. 2006), Hoeffding’s bound based Drift Detection Methods (HDDM_{A-test} and HDDM_{W-test}) (Frías-Blanco et al. 2015), Adaptive Windowing (ADWIN) (Bifet and Gavaldà 2007), SeqDrift2 (Pears et al. 2014), Fast Hoeffding Drift Detection Method (FHDDM) (Pesaranghader and Viktor 2016), and our new Stacking Fast Hoeffding Drift Detection Methods (FHDDMS), which is introduced in Sect. 5.

As shown in Fig. 2, STREAM READER, CLASSIFIERS and DETECTORS, PAIRS OF DETECTORS AND DETECTORS, and CAR CALCULATOR are the main components of the framework. The input is constituted of a Stream, pairs of classifiers and detectors, and a weight vector.

² The *imax* is a function that finds the index of the pair presenting the highest score.

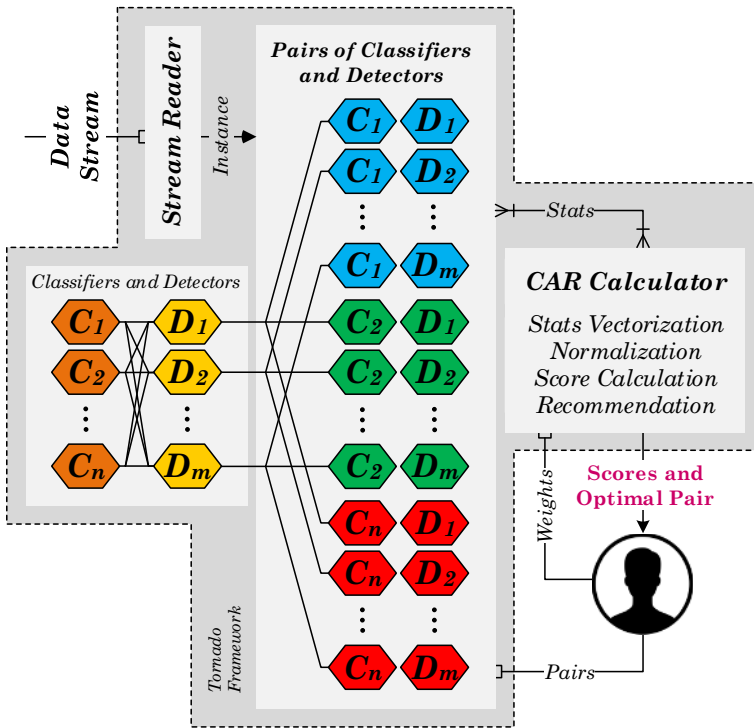


Fig. 2 The TORNADO framework

Our framework follows the *prequential* approach where instances are first tested and then used for training (Gama et al. 2013, 2014; Pesaranghader and Viktor 2016).

The data flow may be described as follows: The (classifier, detector) pairs are constructed as shown in Fig. 2, prior to the learning process. The STREAM READER reads instances from the stream and sends them one-by-one to the (classifier, detector) pairs for model construction. Each learner builds an incremental model, prequentially. That is, each instance is first used for testing and then for training. Simultaneously, CLASSIFIERS send their statistics, e.g. error-rates or the current prediction results, to their corresponding DRIFT DETECTOR in order to detect potential occurrences of concept drifts. Subsequently, the CAR CALCULATOR determines the score of each (classifier, detector) pair by considering the classification error-rate, detection delay, detection false positive rate, detection false negative rate, total memory usage and runtime. Subsequently, the model with the highest score is presented to the user. This model may change as a result of incremental learning and concept drift. This process continues until either a predefined condition is met or all the instances in the stream are processed.

Figure 3 illustrates that, while various pairs are executed concurrently, the one with the best score is recommended at each time interval. An interval is the time difference in between two consecutive concept drifts. This notion is also referred to as *context* in the literature (Gama et al. 2004; Pesaranghader and Viktor 2016). As illustrated by the figure, during the interval τ_0 to τ_n , the pair (C_1, D_3) has the best score. Suddenly, at time τ_n , the data distribution is altered resulting in pair (C_3, D_2) being recommended to the user. In the illustrative example, another drift occurs at τ_{2n} resulting in pair (C_2, D_1) having the highest score.

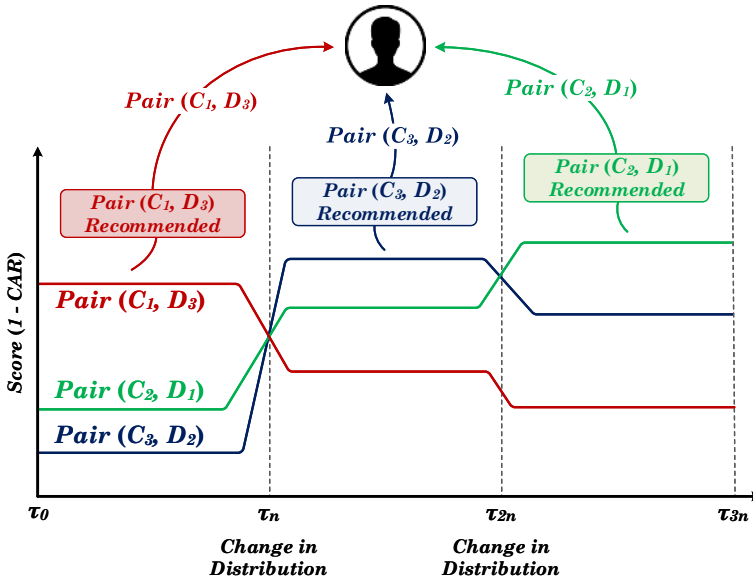


Fig. 3 Recommendation of (Classifier, Detector) Pairs over Time

The following observation is noteworthy. Recall that the continuous outputs of our TORNADO framework are the current best performing (classifier, detector) pair, which may change over time. Consequently, our work should thus not be confused with a hybrid ensemble of classifiers setting (Hsu 2017; Min and Cho 2011; Verikas et al. 2010; Salgado et al. 2006). Typically, a hybrid ensemble contains a number of diverse classifiers that form a committee, which aims at increasing the predictive accuracy by utilizing the diversity of the members of the ensemble. In contrast, within the TORNADO framework, the individual learners proceed independently to construct their models. Recall that the rationale behind our design is that we aim to utilize diverse learning strategies that potentially address concept drifts more efficiently. However, future work may also involve incorporating ensembles, as one of our classifiers, into the TORNADO framework.

5 Stacking fast hoeffding drift detection methods

In this section, we briefly review the Fast Hoeffding Drift Detection Method (FHDDM). We subsequently introduce our Stacking Fast Hoeffding Drift Detection Method (FHDDMS) and Additive FHDDMS (FHDDMS_{add}) algorithm.

5.1 Fast hoeffding drift detection method (FHDDM)

Recently, Pesaraghader and Viktor (2016) introduced the Fast Hoeffding Drift Detection Method (FHDDM) which is based on a sliding window mechanism and the Hoeffding inequality. The FHDDM algorithm slides a window of size n over the classification results. A 1 is inserted in the window if a particular prediction is correct, while a 0 is inserted otherwise. As the instances are processed, the mean associated with a particular sliding window at time

t , μ^t , is evaluated while the maximum mean observed so far, μ^m , is updated if the mean of the current sliding window is higher.

On the basis of the probably approximately correct (PAC) learning model (Mitchell 1997), the classification accuracy either increases or remains constant as the number of instances increases (Gama et al. 2004). Should this not be the case, the probability of a concept drift increases. As a result, the value of μ^m either increases or remains constant as instances are processed. Therefore, a concept drift is more likely if the value of μ^m remains approximately constant while the value of μ^t decreases over time. As demonstrated by Pesaranghader and Viktor (2016), if the difference in between the maximum and the current mean is greater than a certain threshold ε_d , it may be safely assumed that a concept drift has occurred. The threshold is evaluated with the Hoeffding inequality.

Theorem I Hoeffding’s Inequality—*Let X_1, X_2, \dots, X_n be n independent random variables bounded by the interval $[0, 1]$, then with a probability of at most δ , the difference in between the empirical mean of these variables $\bar{X} = \frac{1}{n} \sum_{i=1}^n X_i$ and their expected values $E[\bar{X}]$ is at least ε_H , i.e. $Pr(|\bar{X} - E[\bar{X}]| \geq \varepsilon_H) \leq \delta$, where:*

$$\varepsilon_H = \sqrt{\frac{1}{2n} \ln \frac{2}{\delta}} \tag{5.1}$$

and δ is the upper bound for the probability.

Corollary I FHDDM test—*In a stream setting, assume μ^t is the mean of a sequence of n random entries, where the prediction status of each instance is represented by a value in the set $\{0, 1\}$, at time t . Let μ^m be the maximum mean observed so far. Let $\Delta\mu = \mu^m - \mu^t \geq 0$ be the difference between the two means. Given the desired δ , Hoeffding’s inequality implies that a drift has occurred if $\Delta\mu \geq \varepsilon_d$, where:*

$$\varepsilon_d = \sqrt{\frac{1}{2n} \ln \frac{1}{\delta}} \tag{5.2}$$

Figure 4 illustrates the FHDDM algorithm. In this example, n and δ are set to 10 and 0.2, respectively. Using Corollary I, the value of ε_d is equal to 0.28. Suppose that a real drift occurs right after the 12th instance. The values of μ^t and μ^m are set to null and zero until 10 elements are inserted into the window. We have seven 1s in the window after reading the first 10 elements. Thus μ^t is equal to 0.7 and the value of μ^m is also set to 0.7. The 1st element is removed from the window before the 11th prediction status is inserted. Since the value of prediction status is 0, the value of μ^t decreases to 0.6 while the value of μ^m remains the same. This process continues until the 18th instance is inserted. At this point in time, the difference between μ^m and μ^t exceeds ε_d . As a result, the FHDDM algorithm alarms for a drift.

5.2 Sensitivity of FHDDM’s parameters

In this section, we investigate the impact of the change in parameters δ and n on the value of ε_d . We also study the effects of varying these values on the detection delay, the false positive rate, the memory usage, and total runtime. To this end, we conducted a number of experiments with the values of n in $\{25, 100, 200, 300, 400, 500\}$ and δ in $\{0.001, 0.0001, 0.00001, 0.000001, 0.0000001\}$.

All the results that are shown are against the SINE1 synthetic data stream, which is susceptible to abrupt drift. The classification is the classic $y = \sin(x)$ function and the classes

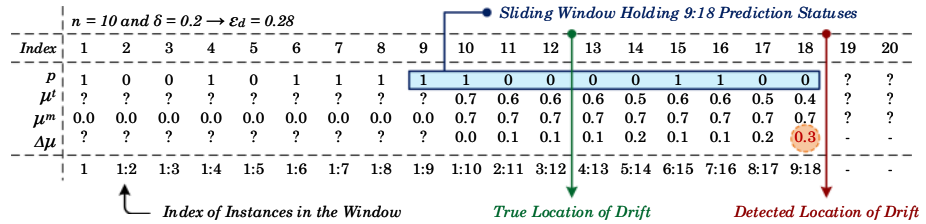


Fig. 4 An example of the FHDDM algorithm

Table 1 Values of FHDDM’s ϵ_d for different n and δ

		δ				
		0.001	0.0001	0.00001	0.000001	0.0000001
n	25	0.37169	0.42919	0.47985	0.52565	0.56777
	100	0.18585	0.21460	0.23993	0.26283	0.28388
	200	0.13141	0.15174	0.16965	0.18585	0.20074
	300	0.10730	0.12390	0.13852	0.15174	0.16390
	400	0.09292	0.10730	0.11996	0.13141	0.14194
	500	0.08311	0.09597	0.10730	0.11754	0.12696

Table 2 Behavior of FHDDM for different values of n , and $\delta = 10^{-7}$

		Delay	TP	FP	FN	Mem.	Average runtime	Total runtime	Error-rate
n	25	38.75	4	0	0	672	55.67	277.71	14.31
	100	49.0	4	0	0	1000	62.69	313.07	14.32
	200	62.5	4	0	0	1376	73.84	367.23	14.38
	300	65.75	4	0	0	1808	81.22	404.30	14.39
	400	72.25	4	0	0	2192	91.82	458.50	14.40
	500	80.25	4	0	0	2680	99.62	495.50	14.42

are reversed at drift points. (Note that more details will be provided in Sect. 6.1.1) Our explorative results are summarized in Tables 1, 2, and 3, as well as in Fig. 5. Note that the averaged runtimes reported are the average of execution runtimes over five contexts. Recall that a context refers to the duration between two consecutive concept drifts. Table 1 shows that, as the value of n increases, the value of ϵ_d decreases. This implies that, since we have more observations, a more optimistic error bound may be used. For a constant n , there is an inverse relationship between δ and ϵ_d . That is, as the value of δ decreases the ϵ_d value increases (i.e. the bound becomes more conservative). Further, Table 2 illustrates that, for a constant value of $\delta = 10^{-7}$, the detection delay increases as we increase the value of n . Intuitively, memory usage and runtime also increase as the window size grows. Table 3 lists the results for a constant $n = 100$. The table shows that, as we decrease the value of δ , the detection delay increases but the false positive rate decreases.

Finally, in Fig. 5, we contrast the memory usage of FHDDM with ADWIN, which also employs a windowing schema. The reader should notice that FHDDM constantly outperforms ADWIN (indicated in red), in terms of memory usage.

Table 3 Behavior of FHDDM for $n = 100$, and different values of δ

		Delay	TP	FP	FN	Mem.	Average runtime	Total runtime	Error-rate
δ	0.001	36.75	4	17	0	1000	14.5	312.46	15.34
	0.0001	44.75	4	5	0	1000	26.71	299.69	14.55
	0.00001	42.75	4	0	0	1000	60.48	301.19	14.31
	0.000001	46.75	4	0	0	1000	61.13	304.49	14.32
	0.0000001	49	4	0	0	1000	60.23	300.25	14.32

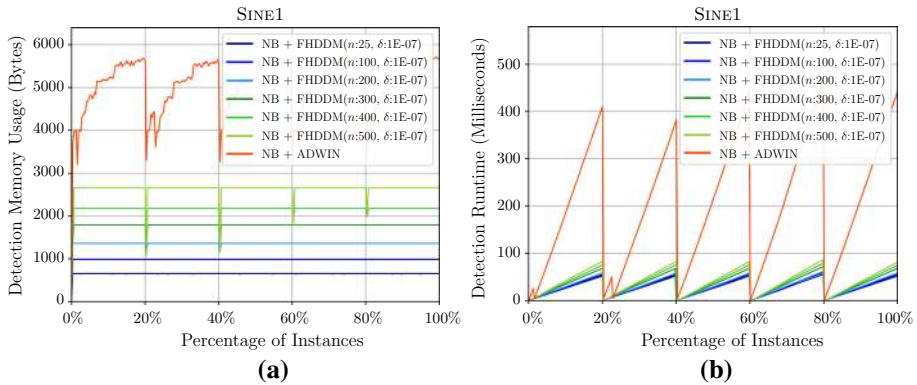


Fig. 5 FHDDM’s memory usage and runtime for different window sizes. **a** Memory Usage, **b** Runtime

Our experiments as reported in Pesaranghader and Viktor (2016) indicated that FHDDM outperformed the state-of-the-art both in terms of adaptation and classification results. It follows that the size of the sliding window is a crucial parameter, as we illustrated above. Our experimental results further indicated that a longer window implies a longer detection delay against abrupt concept drifts. On the other hand, a shorter window may cause higher false negative rates against gradual concept drifts. Based on this observation, we extended our approach as explained in the next section.

5.3 Stacking fast hoeffding drift detection method (FHDDMS)

The Stacking Hoeffding Drift Detection Method (FHDDMS),³ extends the FHDDM method by maintaining windows of different sizes. That is, a short and a long sliding windows are superimposed, as shown in Fig. 6. The rationale behind this approach is to reduce the detection delay and false negative rate. Intuitively, a short window should detect abrupt drifts faster, while a long window should detect gradual drifts with a lower false negative rate.

Following FHDDM, the FHDDMS algorithm inserts a 1 into both the short and the long windows when the prediction result is *correct*, whereas a 0 is inserted otherwise. In Fig. 6, which illustrates our approach, the size of the long and the short sliding windows are set to 20 and 5, respectively.

As instances are processed, FHDDMS calculates the means of the elements inside the long and short sliding windows at time t , i.e. μ_l^t and μ_s^t , as the stream is processed. We define

³ We added the ‘S’ at the end of FHDDMS in order to list it behind FHDDM when drift detectors are *alphabetically* ordered.



Fig. 6 Stacking fast hoeffding drift detection method (FHDDMS)

μ_l^m and μ_s^m as the maximum means so far for the long and the short windows, respectively:

$$\begin{aligned} \mu_l^m < \mu_l^t &\Rightarrow \mu_l^m = \mu_l^t \\ \mu_s^m < \mu_s^t &\Rightarrow \mu_s^m = \mu_s^t \end{aligned} \tag{5.3}$$

Recall that, as based on the probably approximately correct (PAC) learning model (Mitchell 1997), the classification accuracy increases or remains constant as the number of instances increases. Otherwise, the possibility of facing concept drift increases (Gama et al. 2004; Pesaranghader and Viktor 2016). Thus, both the values of μ_l^m and μ_s^m should increase or remain constant as we process instances. Alternatively, the probability of a drift increases if the values of μ_l^m and μ_s^m do not change and the values of μ_s^t and μ_s^t decrease over time. As shown in Eq. (5.4), a significant difference between the current means and their maximums indicates the occurrence of a drift in the stream.

$$\begin{aligned} \Delta\mu_l = \mu_l^m - \mu_l^t \geq \varepsilon_l &\Rightarrow \psi_l = True \\ \Delta\mu_s = \mu_s^m - \mu_s^t \geq \varepsilon_s &\Rightarrow \psi_s = True \\ \text{if } (\psi_l = True) \text{ or } (\psi_s = True) &\Rightarrow \text{alarm for a drift} \end{aligned} \tag{5.4}$$

Here ψ_l and ψ_s denote the status of concept drift detection as observed by the long and short windows, respectively.

Following (Pesaranghader and Viktor 2016), we use the previously introduced Hoeffding’s inequality (Hoeffding 1963) and Corollary I to define the values of ε_l and ε_s :

$$\varepsilon_l = \sqrt{\frac{1}{2|W_l|} \ln \frac{1}{\delta}} \text{ and } \varepsilon_s = \sqrt{\frac{1}{2|W_s|} \ln \frac{1}{\delta}}, \tag{5.5}$$

where $|W_l|$ and $|W_s|$ are the sizes of the long and the short windows, respectively.

The pseudocode for the FHDDMS algorithm is presented in Algorithm 1. The INITIALIZE function initializes the parameters for the stacking windows. Subsequently, while data stream instances are prequentially processed, the DETECT function analyses the prediction results in order to determine if a concept drift has occurred (lines 24–27). A drift is either detected when $(\mu_l^m - \mu_l^t) \geq \varepsilon_l$ or when $(\mu_s^m - \mu_s^t) \geq \varepsilon_s$.

5.4 Additive FHDDMS (FHDDMS_{add})

We introduce another version of the FHDDMS method called as the Additive FHDDMS, i.e. denoted as FHDDMS_{add}. In this approach, the binary indicators are substituted by their summations. As shown in Fig. 7, the short and the long windows are characterized by the sum of their respective most recent prediction results. In this example, the short window holds a single summation of the 5 most recent bits, while the long window holds four summations for the 20 most recent bits seen so far. For each window, the maximum values observed so far, which are μ_s^m and μ_l^m , are updated as required. In this example, the mean values μ_l^t and μ_s^t , are 0.8 and 0.6, respectively. As for the FHDDMS algorithm, a concept drift occurs if the

Algorithm 1 Pseudocode of FHDDMS

```

1: function INITIALIZE( $|W_l|, |W_s|, \delta$ )
2:    $n_l = |W_l|$  ▷ The size of the long window.
3:    $n_s = |W_s|$  ▷ The size of the short window.
4:    $\delta = \delta$ 
5:    $\epsilon_l = \sqrt{\frac{1}{2n_l} \ln \frac{1}{\delta}}, \epsilon_s = \sqrt{\frac{1}{2n_s} \ln \frac{1}{\delta}}$ 
6:   RESET()
7: end function

8: function RESET() ▷ Creating an empty sliding window for stacking.
9:    $Win = []$ 
10:   $\mu_l^m, \mu_s^m = 0$ 
11: end function

12: function DETECT( $p$ ) ▷  $p$  is 1 if the correct predictions, 0 otherwise.
13:  if  $Win$  is full then
14:    drop an element from tail
15:  end if
16:  insert  $p$  into  $Win$ 
17:  calculate  $\mu_l^t$  and  $\mu_s^t$ 
18:  if  $\mu_l^m < \mu_l^t$  then
19:     $\mu_l^m = \mu_l^t$ 
20:  end if
21:  if  $\mu_s^m < \mu_s^t$  then
22:     $\mu_s^m = \mu_s^t$ 
23:  end if
24:  if  $(\mu_l^m - \mu_l^t) \geq \epsilon_l$  or  $(\mu_s^m - \mu_s^t) \geq \epsilon_s$  then
25:    RESET() ▷ Resetting parameters.
26:    return True ▷ Signaling for an alarm.
27:  end if
28:  return False
29: end function

```

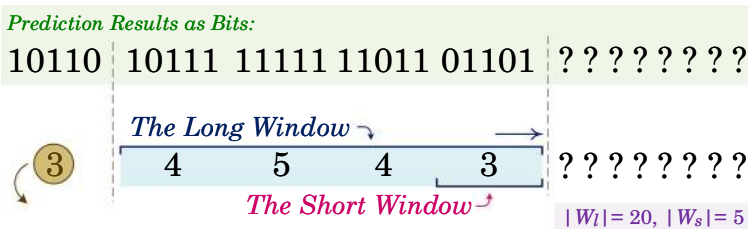


Fig. 7 Additive stacking windows (FHDDMS_{add}) Approach

difference in between the current values and the maximum values exceed a certain threshold, as determined by Hoeffding’s inequality.

Note that, intuitively, FHDDMS_{add} should require less memory and exhibit a faster execution time when compared to FHDDMS, since the data structure is more concise. Nevertheless, this may lead to a longer drift detection delay, since the algorithm must ensure that the short window has accumulated $|W_s|$ new predictions, after an element has been removed from the long window’s tail. This is further confirmed by our experimental results in Sect. 6.3.

Figure 8 illustrates how the FHDDMS and FHDDMS_{add} algorithms proceed. In this toy example, the sizes of the long and short windows are set to 20 and 5, respectively. We also set δ to 0.002. Using Eq. (5.5), we have ϵ_l equal to 0.394, and ϵ_s to 0.788. Recall that FHDDMS

$$\begin{matrix} |W_l| = 20 \\ |W_s| = 5 \end{matrix}, \delta = 0.002 \rightarrow \begin{matrix} \epsilon_l = 0.394 \\ \epsilon_s = 0.788 \end{matrix}$$

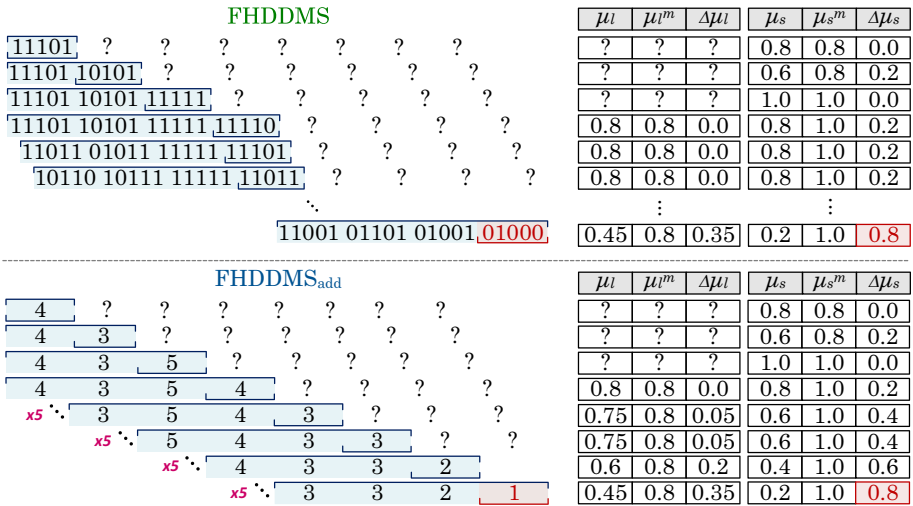


Fig. 8 Example of FHDDMS and FHDDMS_{add} algorithms

considers the predictions bit-by-bit; whereas, FHDDMS_{add} calculates the summary statistics of every 5 predictions. Throughout this incremental learning process, the values of $\mu_l, \mu_l^m, \Delta\mu_l, \mu_s, \mu_s^m, \Delta\mu_s$ are continuously updated, as indicated in the right-side of the illustration. The reader may notice that both algorithms alarm for concept drift when $\Delta\mu_s$ exceeds 0.8, i.e. has a value greater than ϵ_s .

Finally, the theoretical proofs on the bounds of false positive and false negative for the Fast Hoeffding Drift Detection Methods, including FHDDM and FHDDMS, are available in Electronic supplementary material.

6 Experimental evaluation

In this section, we evaluate our new drift detection methods FHDDMS and FHDDMS_{add}, by comparing them against the state-of-the-art. Subsequently, we perform various experiments utilizing the TORNADO framework. We generated synthetic data streams and also considered real-world data for our experiments. We describe the synthetic and real-world data streams as well as the experimental setup in Sects. 6.1 and 6.2 respectively. We evaluate our drift detection methods and the TORNADO framework in Sects. 6.3 and 6.5. Our experiments are performed with a processor Intel Core i5 @ 2 × 2.30 GHz with 16 GB of RAM.

6.1 Data streams used in experimentation

6.1.1 Synthetic data streams

We have selected the previously introduced SINE1 data stream, as well as the SINE2, MIXED, STAGGER, CIRCLES and LED streams, which are frequently applied in the data stream mining literature (Kubat and Widmer 1995; Nishida and Yamauchi 2007; Bifet and Gavaldà 2007;

Table 4 Summary of synthetic data streams

Data stream	Attribute	Attr. type	Class	Drift points	ζ	Noise (%)	Drift type
SINE1	2	Numeric	2	$\times 20,000$	50	10	Abrupt
SINE2	2	Numeric	2	$\times 20,000$	50	10	Abrupt
MIXED	4	Mixed	2	$\times 20,000$	50	10	Abrupt
STAGGER	3	Nominal	2	$\times 33,333$	50	10	Abrupt
CIRCLES	2	Numeric	2	$\times 25,000$	500	10	Gradual
LED	24	{0, 1}	10	$\times 25,000$	500	10	Gradual

Frías-Blanco et al. 2015; Gama et al. 2004; Olorunnimbe et al. 2015; Pesaranghader et al. 2016; Pesaranghader and Viktor 2016; Barros et al. 2017, 2018,) as the synthetic data streams for our experiments. Each data stream contains 100,000 instances. SINE1, SINE2, MIXED, STAGGER, CIRCLES have only two class labels, whereas LED has 10 class labels. Following the convention, we have placed drift points at every 20,000 instances in SINE1, SINE2, and MIXED, and at every 33,333 instances in STAGGER with a transition length of $\zeta = 50$ to simulate *abrupt* concept drift. In addition, we have put drift points at every 25,000 instances in CIRCLES and LED data streams with a transition length of $\zeta = 500$ to simulate *gradual* concept drift. We have added 10% noise to each data stream, as well, to observe how robust drift detectors are against noisy data streams by asserting their ability to distinguish between concept drift and noise. Table 4 summarizes the synthetic data streams. They may be described as follow:

- SINE1 · *with abrupt drift*: Recall that the stream consists of two attributes x and y uniformly distributed in the interval $[0, 1]$. The classification function is $y = \sin(x)$. Instances are classified as positive if they are under the curve; otherwise they are classified as negative. At a drift point, the class labels are reversed.
- SINE2 · *with abrupt concept drift*: It holds two attributes x and y which are uniformly distributed in between 0 and 1. The classification function is $0.5 + 0.3 * \sin(3\pi x)$. Instances under the curve are classified as positive while the remaining instances are classified as negative. At a drift point, the classification scheme is inverted.
- MIXED · *with abrupt drift*: The dataset has two numeric attributes x and y distributed in the interval $[0, 1]$ as well as two boolean attributes v and w . The instances are classified as positive if at least two of the following three conditions are satisfied: $v, w, y < 0.5 + 0.3 * \sin(3\pi x)$. The classification is reversed when drift points occur.
- STAGGER · *with abrupt concept drift*: This dataset contains three nominal attributes, namely *size* {small, medium, large}, *color* {red, green} and *shape* {circular, non-circular}. Before the first drift point, instances are labeled positive if $(color = red) \wedge (size = small)$. After this point and before the second drift, instances are classified positive if $(color = green) \vee (shape = circular)$, and finally after the second drift point, instances are classified positive only if $(size = medium) \vee (size = large)$.
- CIRCLES · *with gradual drift*: This dataset contains two attributes x and y which are uniformly distributed in the interval $[0, 1]$. The classification function is $(x - x_c)^2 + (y - y_c)^2 = r_c^2$ where (x_c, y_c) is its center and r_c is the radius. Instances inside the circle are classified as positive. A drift happens whenever the classification function, i.e. the circle function, changes.

- LED · *with gradual drift*: The objective of this dataset is to predict the digit on a seven-segment display, where each digit has a 10% chance of being displayed. The dataset has 7 attributes related to the class, and 17 irrelevant ones. Concept drift is simulated by interchanging relevant attributes (Frías-Blanco et al. 2015).

6.1.2 Real-world data streams

We further conducted experiments using the following real-world data streams⁴; which are frequently used in the online learning and adaptive learning literature (Gama et al. 2004; Baena-García et al 2006; Bifet et al. 2009; Frías-Blanco et al. 2015). These three data streams were used in our comparative evaluation of drift detectors.

- ELECTRICITY contains 45,312 instances, with 8 input attributes, recorded every half an hour for two years from Australian New South Wales Electricity. The classification task is to predict a rise (*Up*) or a fall (*Down*) in the electricity price. The concept drift may happen because of changes in consumption habits, unexpected events, and seasonality (Žliobaite 2013).
- FOREST COVERTYPE has 54 attributes with 581,012 instances describing 7 forest cover types for 30×30 meter cells obtained from US Forest Service (USFS) Region 2 Resource Information System (RIS) data, for four wilderness areas located in the Roosevelt National Forest of northern Colorado (Blackard and Dean 1999).
- POKER HAND consists of 1,000,000 instances, where each instance is an example of a hand having five playing cards drawn from a standard deck of 52. Each card is described by two attributes (suit and rank), for ten predictive attributes. The class predicts the poker hand (Olorunnimbe et al. 2015).

To the best of our knowledge, there are no real-world datasets publicly available, wherein the locations of concept drifts are clearly identified. For instance, there is consensus among researchers that the location and/or presence of concept drift in the ELECTRICITY, FOREST COVERTYPE, and POKERHAND data stream are unknown (Huang et al. 2015; Bifet and Gavaldà 2007; Pesaranghader and Viktor 2016; Frías-Blanco et al. 2015; Bifet et al. 2009). Therefore, in addition to the data streams mentioned above, we also used *static* datasets, publicly available from the UCI machine learning repository (Bache and Lichman 2013), and we simulated concept drift by switching labels at drift points. We considered the ADULT (Kohavi 1996), NURSERY (Zupan et al. 1997), and SHUTTLE (Catlett 2002) datasets for our study. We describe the original datasets as well as the preprocessing steps adopted to generate the corresponding data streams below:

- ADULT: The original dataset has six numeric and eight nominal attributes, two class labels, and 48,842 instances out of which 32,561 instances are used for building the classification models. The dataset was used to predict whether a person earns an annual income greater than \$50,000 (Kohavi 1996).
 - ▷ *Preprocessing*: The training dataset is imbalanced, and there are 24,720 instances for class $\leq 50K$ as oppose to 7841 instances for class $> 50K$. We first undersampled the data, leading to 8200 instances for class $\leq 50K$ and 7800 instances for class $> 50K$. We subsequently increased the number of instances to 20,000 by bootstrapping.
- NURSERY: The dataset holds eight nominal attributes, five class labels, and 12,960 instances. It was designed to predict whether applications for nursery schools in Ljubljana, Slovenia should be rejected or accepted (Zupan et al. 1997).

⁴ Available at: <http://moa.cms.waikato.ac.nz/datasets/2013/>.

- ▷ *Preprocessing*: The dataset consists of 5 classes labelled as ‘no_recom’, ‘recommend’, ‘very_recom’, ‘priority’, and ‘spec_priority’. The number of occurrences of the third and fourth classes are infrequent and we removed them from the dataset, resulting in a dataset consisting of 20,000 instances.
- SHUTTLE: The original dataset contains nine attributes, seven class labels, and 58,000 instances and was designed to predict suspicious states during a NASA shuttle mission (Catlett 2002).
 - ▷ *Preprocessing*: This dataset is also highly imbalanced. Firstly, instances from the four minority classes were filtered out and undersampling and bootstrapping were performed, in order to create a dataset of 20,000 instances.

Finally, we simulated concept drift by shifting the class labels after drift points, with a transition length of $\zeta = 50$, for the new context. Note that we use the term ‘context’ to refer to the interval between two consecutive concept drifts. The final data streams have five contexts, each including 20,000 instances, for 100,000 instances in total.⁵

6.2 Experimental setting

Following Bifet et al. (2009), we used the sigmoid function to simulate abrupt and gradual concept drifts. The function determines the probability of belonging to the new context during the transition between two contexts. The transition length ζ allows us to simulate abrupt or gradual concept drifts. It is set to 50 for abrupt concept drifts, and to 500 for gradual concept drifts in all our experiments.

Pesaranghader and Viktor (2016) proposed an approach to evaluate drift detection methods. They introduced the *acceptable delay length* notion to count true positive (TP), false positive (FP), and false negative (FN) rates. The acceptable delay length Δ is a threshold that determines how far the detected drift could be from its true location for the drift to be considered as true positive (Pesaranghader and Viktor 2016; Krawczyk et al. 2017). That is, we maintain three variables to count the numbers of true positives, false negatives and false positives. These variables are initially set to zero. We increment the number of true positives when the drift detector alarm is within the acceptable delay range. Otherwise, we increment the number of false negatives, since the alarm has occurred too late. In addition, the false positive value is incremented when a false alarm occurs, outside of the acceptable delay range. Following this approach, we set Δ to 250 for the SINE1, SINE2, MIXED, STAGGER, and the real world-world data streams, and to 1000 for the CIRCLES and LED data streams. A longer Δ should be considered for data streams with gradual drifts in order to avoid an increase in false negatives (Pesaranghader and Viktor 2016).

Finally, for FHDDMS_{add} and FHDDMS, the size of the long window and the short window are set to 100 and 25, respectively. The parameter δ is set to 10^{-7} in all cases. Recall that, as for the other drift detectors, all parameters were set to default values.

6.3 Evaluation of FHDDMS and FHDDMS_{add} drift detectors

In these experiments, our aim is to assess the capabilities of FHDDMS and FHDDMS_{add} to detect concept drifts in synthetic as well as in real-world data streams. Firstly, our goal is to determine whether our algorithms are able to detect change fast, while maintaining high accuracies, as measured when considering true positive, false positive, and false negative rates. Secondly, we compare our FHDDMS and FHDDMS_{add} methods to the state-of-the-art.

⁵ The data streams and source codes are available at: <http://www.github.com/alipsgH>.

Thirdly, we assess the abilities of detection algorithms to detect drifts without applying a classification algorithm.

6.3.1 Experimental evaluation on synthetic data streams

We compare the performances of FHDDMS and FHDDMS_{add} against DDM, EDDM, HDDMs, CUSUM, Page-Hinkley, ADWIN, SeqDrift2 and FHDDM. We considered Naive Bayes (NB) and Hoeffding Tree (HT) as our incremental learners. We ran each classifier-detector pair 100 times. Recall that we maintained true positive, false positive, and false negative counters for each run, by considering the corresponding *acceptable delay length* Δ of data stream. We captured the memory usage of drift detectors after each alarm, and then averaged them once all instances are processed. The overall detection runtime of drift detectors as well as the overall error-rates classifiers for each run were computed. Please note that the memory usage and the runtime are recorded in *bytes* and *milliseconds*, respectively. Further, we averaged the detection delays, true positives, false positives, false negatives, total detection runtimes, and memory usage of drift detectors as well as the error-rates of the classifiers over all iterations.

I. Drift Detection with Naive Bayes—Tables 5, 6 and 7 summarize the experimental results for Naive Bayes (NB) with all drift detectors. As indicated in Tables 5 and 6, HDDM_{W-test} and FHDDMS have the shortest detection delays, followed by FHDDM₁₀₀ and FHDDMS_{add}. On the other hand, EDDM yields the longest delay before detection, followed by PageHinkley and SeqDrift2 for SINE1 and MIXED as well as DDM for CIRCLES. HDDM_{W-test} has shorter detection delays compared to FHDDMS against abrupt concept drifts, they yield similar detection delays for the CIRCLES and LED data streams containing gradual concept drifts (Table 6). Overall, FHDDMS_{add}, FHDDMS, FHDDM₂₅, FHDDMS₁₀₀, and CUSUM have the lowest false positive rates. ADWIN and SeqDrift2 have a large number of false positive numbers when they are used in conjunction with the LED data stream. Since EDDM does not find concept drifts within the acceptable delay lengths, it resulted in the highest false negative numbers. FHDDMS_{add}, FHDDMS, FHDDMs, HDDMs, and CUSUM have the lowest false negative rates. FHDDMS_{add} outperforms FHDDMS and FHDDM₁₀₀ in terms of memory consumption and detection runtime. ADWIN and SeqDrift2 require more memory than the other approaches.

Finally, as shown in Table 7, we obtained the lowest classification error-rates with FHDDMS and FHDDMs. In general, the classification error-rates are comparable for all data streams, except for LED, where ADWIN and SeqDrift2 have much higher error-rates.

II. Drift Detection with Hoeffding Trees—The experimental results when using Hoeffding Trees with drift detectors are shown in Tables A.1–A.3. Considering both Tables A.1 and A.2, we observe that FHDDMS, FHDDMs, and HDDM_{W-test} have the shortest drift detection delay. HDDM_{W-test} has the shortest detection delay when the concept drifts are abrupt; whereas, FHDDMS has the shortest detection delays when the concept drifts are gradual. FHDDMS, FHDDMs, CUSUM, and DDM caused the lowest false positives of all drift detectors. The false positive numbers of HDDMs are consistently higher than those of FHDDMS and FHDDMs. EDDM, again, resulted in the highest false negative rates.

One should notice that the false positives are more common with Hoeffding Tree than with Naive Bayes. This indicates Hoeffding Tree may not represent decision boundaries adequately, which misleads drift detection methods and consequently causes more false alarms. As Table A.3 depicts, FHDDMS and FHDDMs led to the lowest classification error-rates. In general, the classification error-rates are similar for all data streams, except for LED where ADWIN and SeqDrift2 again resulted in higher error-rates.

Table 5 Naive Bayes classifier and drift detectors against SYNTHETIC data streams with abrupt concept drifts

Naive Bayes—Abrupt concept drift ($\zeta = 50$)						
Detector	Delay	TP	FP	FN	Mem.	Runtime
SINE1						
FHDDMS _{add}	52.06 ± 3.86	4.0 ± 0.0	0.01 ± 0.1	0.0 ± 0.0	880.0	241.89 ± 52.41
FHDDMS	40.52 ± 3.55	4.0 ± 0.0	0.06 ± 0.24	0.0 ± 0.0	1096.0	1087.12 ± 125.51
FHDDM ₂₅	40.87 ± 3.62	4.0 ± 0.0	0.01 ± 0.1	0.0 ± 0.0	672.0	240.33 ± 53.85
FHDDM ₁₀₀	48.48 ± 2.87	4.0 ± 0.0	0.05 ± 0.22	0.0 ± 0.0	1000.0	280.49 ± 60.86
CUSUM	85.14 ± 5.44	4.0 ± 0.0	0.03 ± 0.17	0.0 ± 0.0	544.0	287.36 ± 2.47
PageHinkley	234.47 ± 11.7	1.74 ± 0.99	2.26 ± 0.99	2.26 ± 0.99	592.0	232.42 ± 1.97
DDM	152.89 ± 9.18	3.99 ± 0.1	0.01 ± 0.1	0.01 ± 0.1	576.0	332.21 ± 78.49
EDDM	249.42 ± 5.72	0.01 ± 0.1	7.53 ± 2.68	3.99 ± 0.1	920.0	140.79 ± 43.16
ADWIN	65.63 ± 2.54	4.0 ± 0.0	4.08 ± 2.4	0.0 ± 0.0	> 4310	2191.71 ± 141.98
SeqDrift2	200.83 ± 0.89	4.0 ± 0.0	1.74 ± 1.51	0.0 ± 0.0	6616.0	888.21 ± 97.74
HDDM _{A-test}	68.33 ± 16.09	3.99 ± 0.1	0.43 ± 0.65	0.01 ± 0.1	656.0	1122.68 ± 8.17
HDDM _{W-test}	32.97 ± 3.28	4.0 ± 0.0	0.5 ± 0.71	0.0 ± 0.0	1504.0	1134.65 ± 5.14
MIXED						
FHDDMS _{add}	52.19 ± 4.09	4.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	880.0	249.24 ± 68.06
FHDDMS	40.43 ± 3.43	4.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	1096.0	1079.31 ± 123.36
FHDDM ₂₅	40.80 ± 3.45	4.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	672.0	240.65 ± 56.12
FHDDM ₁₀₀	48.44 ± 3.21	4.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	1000.0	281.9 ± 63.47
CUSUM	85.35 ± 4.38	4.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	544.0	292.31 ± 2.91
PageHinkley	241.36 ± 7.84	1.23 ± 0.88	2.77 ± 0.88	2.77 ± 0.88	592.0	230.52 ± 1.69
DDM	147.36 ± 6.7	3.99 ± 0.1	0.05 ± 0.22	0.01 ± 0.1	576.0	356.23 ± 73.79
EDDM	250.0 ± 0.0	0.0 ± 0.0	8.2 ± 2.71	4.0 ± 0.0	920.0	148.61 ± 49.39
ADWIN	68.11 ± 8.48	3.97 ± 0.17	9.65 ± 4.12	0.03 ± 0.17	> 4660	2216.46 ± 183.64
SeqDrift2	200.92 ± 1.11	4.0 ± 0.0	1.77 ± 1.71	0.0 ± 0.0	6616.0	925.54 ± 198.3
HDDM _{A-test}	64.85 ± 16.16	4.0 ± 0.0	0.25 ± 0.5	0.0 ± 0.0	656.0	1120.06 ± 8.12
HDDM _{W-test}	33.14 ± 3.31	4.0 ± 0.0	0.34 ± 0.62	0.0 ± 0.0	1504.0	1140.31 ± 10.54

The best-performing methods are given in bold

III. Drift Detection Assessment—Next, we investigate the capabilities of the FHDDMS and FHDDMS_{add} algorithms to detect drifts. Here, our aim is to determine whether the detection algorithms are able to detect change in an accurate manner, without the assistance of a classification algorithm. To this end, we present the results against synthetic data streams containing 50,000 bits, following (Bifet and Gavalda 2007; Huang et al. 2015). In this setting, each stream contains 5 segments, where each segment holds 10,000 bits. We simulated abrupt and gradual concept drifts at the *center* of each segment. That is, in a stream, there exist 5 drifts at locations 5000, 15,000, 25,000, 35,000, and 45,000, respectively. The probability of observing a 1 as a bit before and after a drift is 80% and 20%, respectively. We considered two values of 0.005 and 0.001 for the *change ratio* r , i.e. to simulate *gradual* drift. Here, the change ratio defines the rate of shift from one context to another. The acceptable delay length, i.e. Δ , was set to 5,000 for measuring detection delay, as well as false positive (FP) and false negative (FN) numbers. We conducted a set of preliminary experiments to find

Table 6 Naive Bayes classifier and drift detectors against SYNTHETIC data streams with gradual concept drifts

Naive Bayes—Gradual concept drift ($\zeta = 500$)						
Detector	Delay	TP	FP	FN	Mem.	Runtime
CIRCLES						
FHDDMS _{add}	216.08 ± 110.18	2.84 ± 0.37	0.17 ± 0.4	0.16 ± 0.37	880.0	250.03 ± 59.11
FHDDMS	142.59 ± 78.99	2.97 ± 0.17	0.06 ± 0.24	0.03 ± 0.17	1096.0	1086.49 ± 128.8
FHDDM ₂₅	422.51 ± 96.4	2.22 ± 0.41	0.53 ± 0.5	0.78 ± 0.41	672.0	252.52 ± 66.658
FHDDM ₁₀₀	145.02 ± 78.23	2.97 ± 0.17	0.05 ± 0.22	0.03 ± 0.17	1000.0	281.58 ± 65.11
CUSUM	235.06 ± 45.68	2.99 ± 0.1	0.19 ± 0.39	0.01 ± 0.1	544.0	286.83 ± 2.3
PageHinkley	598.51 ± 62.38	2.58 ± 0.49	0.42 ± 0.49	0.42 ± 0.49	592.0	232.36 ± 1.25
DDM	514.19 ± 63.85	2.68 ± 0.47	0.39 ± 0.53	0.32 ± 0.47	576.0	345.81 ± 62.88
EDDM	960.57 ± 83.79	0.35 ± 0.55	8.19 ± 3.46	2.65 ± 0.55	920.0	136.57 ± 42.4
ADWIN	159.77 ± 35.94	3.0 ± 0.0	1.47 ± 0.82	0.0 ± 0.0	> 4885	2210.83 ± 156.5
SeqDrift2	226.28 ± 44.06	3.0 ± 0.0	0.72 ± 0.9	0.0 ± 0.0	6616.0	871.95 ± 87.94
HDDM _{A-test}	246.41 ± 106.38	2.90 ± 0.3	0.46 ± 0.57	0.1 ± 0.3	656.0	1135.89 ± 5.92
HDDM _{W-test}	141.88 ± 96.14	2.93 ± 0.26	0.56 ± 0.79	0.07 ± 0.26	1504.0	1142.57 ± 18.81
LED _{0.3.1.3}						
FHDDMS _{add}	281.83 ± 72.32	2.99 ± 0.1	0.01 ± 0.1	0.01 ± 0.1	880.0	264.28 ± 75.8
FHDDMS	250.79 ± 55.84	3.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	1096.0	1247.52 ± 155
FHDDM ₂₅	423.84 ± 134.95	2.8 ± 0.53	0.06 ± 0.28	0.2 ± 0.53	672.0	262.21 ± 59.06
FHDDM ₁₀₀	250.79 ± 55.84	3.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	1000.0	260.95 ± 65.26
CUSUM	298.88 ± 50.33	3.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	544.0	287.81 ± 4.98
PageHinkley	563.72 ± 79.48	2.95 ± 0.26	0.04 ± 0.24	0.05 ± 0.26	592.0	223.04 ± 4.14
DDM	443.14 ± 71.08	3.0 ± 0.0	0.01 ± 0.1	0.0 ± 0.0	576.0	374.58 ± 89.38
EDDM	966.75 ± 55.69	0.47 ± 0.56	2.82 ± 0.97	2.53 ± 0.56	920.0	144.49 ± 42.7
ADWIN	554.82 ± 208.41	2.46 ± 0.67	347.14 ± 9.4	0.54 ± 0.67	> 3760	1790.85 ± 161.3
SeqDrift2	469.04 ± 206.09	2.63 ± 0.63	235.5 ± 19.9	0.37 ± 0.63	6616.0	864.03 ± 121.25
HDDM _{A-test}	294.94 ± 82.46	2.99 ± 0.1	0.18 ± 0.5	0.01 ± 0.1	656.0	1099.1 ± 11.59
HDDM _{W-test}	257.21 ± 87.34	2.95 ± 0.26	0.08 ± 0.27	0.05 ± 0.26	1504.0	1098.25 ± 11.26

The best-performing methods are given in bold

unbiased values for the r and Δ parameters. Table 8 summarizes the results of experiments in terms of average detection delay, false positive (FP), and false negative (FN) numbers. The reader should also note that we had 100 streams for each experiments, and we averaged the results for each method. We discuss the results as follows.

Table 8 shows, for the case of abrupt drift, FHDDMS, FHDDM₂₅, and HDDM_{W-test} had the shortest detection delays. On the other hand, PageHinkley and EDDM resulted in the longest detection delays. FHDDMS, and FHDDMs led to the lowest false positive numbers. HDDM_{W-test} had a greater false positive number compared to FHDDMS, FHDDM, and HDDM_{A-test}. Similar to previous experiments, EDDM showed the highest number of false positive. Although PageHinkley showed a small number of false positive, it had the greatest false negative number amongst others. EDDM and PageHinkley also showed high detection delays.

As to gradual drifts, we obtained the shortest detection delays with FHDDMS, FHDDMs and HDDM_{W-test}; nevertheless, HDDM_{W-test} caused greater false positives. FHDDMS_{add}

Table 7 Naive Bayes error-rates against all SYNTHETIC data streams

Naive Bayes error-rate				
Detector	SINE1	MIXED	CIRCLES	LED _{0.3.1.3}
FHDDMS _{add}	14.39 ± 0.17	13.51 ± 0.11	13.88 ± 0.12	10.45 ± 0.03
FHDDMS	14.37 ± 0.17	13.49 ± 0.11	13.83 ± 0.08	10.44 ± 0.04
FHDDM ₂₅	14.37 ± 0.17	13.49 ± 0.11	14.58 ± 0.73	10.52 ± 0.23
FHDDM ₁₀₀	14.38 ± 0.17	13.51 ± 0.11	13.83 ± 0.08	10.44 ± 0.04
CUSUM	14.48 ± 0.17	13.61 ± 0.11	13.88 ± 0.07	10.44 ± 0.03
PageHinkley	14.98 ± 0.18	14.15 ± 0.13	14.08 ± 0.09	10.67 ± 0.04
DDM	14.68 ± 0.17	13.80 ± 0.11	14.04 ± 0.10	10.52 ± 0.02
EDDM	16.97 ± 0.26	16.08 ± 0.19	15.18 ± 0.33	11.67 ± 0.20
ADWIN	14.74 ± 0.23	14.35 ± 0.34	13.85 ± 0.07	27.79 ± 0.56
SeqDrift2	14.88 ± 0.19	14.04 ± 0.14	13.88 ± 0.07	22.58 ± 1.13
HDDM _{A-test}	14.47 ± 0.18	13.58 ± 0.12	13.88 ± 0.09	10.47 ± 0.05
HDDM _{W-test}	14.38 ± 0.18	13.51 ± 0.12	13.83 ± 0.09	10.45 ± 0.04
NO DETECTION	43.01 ± 0.17	43.24 ± 0.14	24.58 ± 0.14	27.4 ± 4.41

The best-performing methods are given in bold

Table 8 Experiments with synthetic bit streams

Detector	Abrupt		
	Delay	FP	FN
FHDDMS _{add}	22.35 ± 1.28	0.01 ± 0.10	0.0 ± 0.0
FHDDMS	13.02 ± 1.55	0.29 ± 0.52	0.0 ± 0.0
FHDDM ₂₅	13.05 ± 1.53	0.19 ± 0.44	0.0 ± 0.0
FHDDM ₁₀₀	27.01 ± 3.0	0.10 ± 0.30	0.0 ± 0.0
CUSUM	153.98 ± 97.32	0.26 ± 0.52	0.01 ± 0.10
PageHinkley	3066.16 ± 29.13	0.01 ± 0.10	1.01 ± 0.10
DDM	301.08 ± 38.78	0.61 ± 0.96	0.0 ± 0.0
EDDM	1074.73 ± 76.60	33.57 ± 11.78	0.0 ± 0.0
ADWIN	42.82 ± 4.71	4.0 ± 0.0	0.0 ± 0.0
SeqDrift2	197.0 ± 0.0	4.11 ± 0.34	0.0 ± 0.0
HDDM _{A-test}	36.31 ± 12.58	0.45 ± 0.59	0.0 ± 0.0
HDDM _{W-test}	18.61 ± 99.49	2.69 ± 1.62	0.01 ± 0.10

Detector	Gradual					
	<i>r</i> = 0.005			<i>r</i> = 0.001		
	Delay	FP	FN	Delay	FP	FN
FHDDMS _{add}	90.45 ± 6.51	0.05 ± 0.22	0.0 ± 0.0	235.35 ± 21.13	4.95 ± 0.33	0.0 ± 0.0
FHDDMS	72.29 ± 6.08	0.41 ± 0.62	0.0 ± 0.0	202.36 ± 18.78	5.44 ± 0.68	0.0 ± 0.0
FHDDM ₂₅	74.98 ± 6.99	0.45 ± 0.68	0.0 ± 0.0	257.46 ± 25.85	4.32 ± 1.02	0.0 ± 0.0
FHDDM ₁₀₀	80.46 ± 5.06	0.11 ± 0.31	0.0 ± 0.0	206.40 ± 18.33	5.23 ± 0.51	0.0 ± 0.0
CUSUM	220.52 ± 11.18	0.32 ± 0.60	0.0 ± 0.0	515.51 ± 21.10	1.32 ± 0.72	0.0 ± 0.0
PageHinkley	3091.74 ± 29.3	0.01 ± 0.10	1.02 ± 0.1	2987.62 ± 288	0.65 ± 0.52	0.78 ± 0.4

Table 8 continued

Detector	Gradual					
	$r = 0.005$			$r = 0.001$		
	Delay	FP	FN	Delay	FP	FN
DDM	375.48 ± 39.0	0.62 ± 1.01	0.0 ± 0.0	677.84 ± 45.56	1.24 ± 1.26	0.0 ± 0.0
EDDM	1124.73 ± 84.8	32.83 ± 12	0.0 ± 0.0	1311.78 ± 105.8	35.28 ± 14.1	0.0 ± 0.0
ADWIN	104.58 ± 6.88	4.0 ± 0.0	0.0 ± 0.0	255.24 ± 15.75	8.76 ± 0.47	0.0 ± 0.0
SeqDrift2	197.0 ± 0.0	4.11 ± 0.34	0.0 ± 0.0	332.60 ± 42.67	6.86 ± 1.18	0.0 ± 0.0
HDDM _{A-test}	97.46 ± 14.13	0.77 ± 0.83	0.0 ± 0.0	249.45 ± 25.32	5.60 ± 1.04	0.0 ± 0.0
HDDM _{W-test}	61.03 ± 6.10	3.71 ± 1.91	0.0 ± 0.0	182.58 ± 23.37	9.01 ± 1.72	0.0 ± 0.0

The best-performing methods are given in bold

and FHDDM₁₀₀ resulted in lower false positive numbers when $r = 0.005$; whereas CUSUM and DDM were more accurate when $r = 0.001$. Similar to the case of abrupt drift, EDDM and PageHinkley had the highest false positive and false negative numbers, respectively. They also detected drifts with lengthy delays. Further, the reader may notice that false positives increased by decreasing the change ratio from 0.005 to 0.001. Indeed, a slower shift makes it harder for all methods to accurately detect drifts. Consequently, we experienced that abundance in the false positives.

In summary, our results confirm the abilities of the FHDDM family of drift detection algorithms to detect drift, in the absence of a classification algorithm.

IV. Conclusion—Recall that the aim of these experiments was threefold. Firstly, we assessed the performance of the FHDDM family of drift detection algorithms. We observed that FHDDMS yield better performances compared to FHDDM₂₅ and FHDDM₁₀₀ against data stream containing both *abrupt* and *gradual* concept drifts. That is, the stacking of sliding windows assisted to detect concept drifts with shorter detection delays and fewer false negatives. Recall that this method slides a short window as well as a long window on prediction results. The short window finds abrupt drifts with shorter delays, while the long window detects gradual drifts with fewer false negatives. FHDDMS_{add} had fewer false positives, less memory usage, and shorter runtime compared to FHDDMS. Secondly, our results confirm that our algorithms performs well, when compared to the state-of-the-art. Although HDDM_{W-test} had similar detection delays compared to FHDDMS and FHDDMs, it resulted in higher false positive rates. Finally, our experimental evaluation show the correctness of our approaches, when detecting drift without the aid of a classification algorithm.

6.4 Experiments on real-world data streams

In this section, we present the results of our experiments on the ELECTRICITY, FOREST COVERTYPE, and POKER HAND data streams, as introduced in Sect. 6.1.2, using Naive Bayes (NB) and Hoeffding Tree (HT) as the incremental learners. Again we stress that, as pointed out by Huang et al. (2015) and Bifet et al. (2009), the locations of concept drifts are not known in these data streams. We therefore follow the work of Huang et al. (2015) and establish our evaluations based on the number of alarms for concept drift and the classification error-rates. Our experimental results are summarized in Table 9 as well as Tables A.4 and A.5.

Since we are not aware of the exact drift locations, we do not make strong conclusions about the performance, in terms of drift detection, of the algorithms. However, when considering

Table 9 Naive Bayes and Hoeff. Tree against ELECTRICITY Data Stream

	ELECTRICITY							
	Memory		Runtime		Num. Drifts		Error-rate	
	NB	HT	NB	HT	NB	HT	NB	HT
FHDDMS _{add}	880.0	880.0	140.41	174.98	68	64	26.86	26.80
FHDDMS	1096.0	1096.0	450.14	582.13	96	102	26.05	26.64
FHDDM ₂₅	672.0	672.0	125.56	168.66	102	102	26.23	26.70
FHDDM ₁₀₀	1000.0	1000.0	141.84	195.5	57	56	26.54	26.38
CUSUM	544.0	544.0	137.98	180.07	21	19	28.35	27.95
PageHinkley	592.0	592.0	112.37	142.8	9	7	29.91	28.64
DDM	576.0	576.0	178.87	231.41	28	9	30.82	29.98
EDDM	920.0	920.0	88.43	118.61	195	168	27.42	27.37
ADWIN	4098.76	4066.15	930.33	1094.65	29	26	28.08	27.67
SeqDrift2	6616.0	6616.0	421.54	511.13	21	23	29.33	28.04
HDDM _{A-test}	656.0	656.0	488.85	590.4	166	160	26.37	26.71
HDDM _{W-test}	1504.0	1504.0	515.76	629.05	159	156	26.09	26.45
NO DETECTION	–	–	–	–	–	–	33.49	29.46

The best-performing methods are given in bold

the Electricity data stream, the reader will notice that our FHDDMS algorithms and the HDDM algorithms obtained the lowest error-rates, when combined with both classifiers. The HDDMs and EDDM algorithms alarmed most often for concept drift, while the FHDDMS algorithms signal for drifts more often than the other remaining techniques. Moreover, the memory usages and runtimes of our methods are comparable to the state-of-the art. Similar observations hold for the Forest Covertype and the Poker Hand data streams as represented in Tables A.4 and A.5.

Overall, this result indicates that there is no single drift detector that outperforms in all settings. The reader should recall that, based on this observation, we introduced the TORNADO framework which is used to constantly recommends the currently best performing (classifier, detector) pair to the user. Next, we discuss our experimental evaluation of the TORNADO framework.

6.5 Experimental evaluation of TORNADO framework

This section presents the experimental results for the TORNADO framework for synthetic and real-world data streams. In these experiments, our aim is to illustrate the advantages of employing multiple classifiers and drift detectors in parallel. Our goal is to confirm that the *best* (classifier, detector) pair varies, as the characteristics of a stream change over time. Further, our objective is to confirm that the *best* pair is highly domain dependent; and we explore whether this observation holds against synthetic and real-world streams.

Five learning algorithms were evaluated, namely incremental Naive Bayes (NB), Decision Stump (DS), Hoeffding Tree (HT), Perceptron (PR), and 5 Nearest Neighbors (5-NN) learners. In addition, the previously introduced drift detection methods, in Sect. 2.2, were paired with the classifiers. As a result, we have a total of 60 pairs of (classifier, detector). The scores associated with the (classifier, detector) pairs are calculated utilizing the CAR measure, as

Pair Recommendation over Time - Abrupt Concept Drift
(from top-left corner to bottom-right corner)

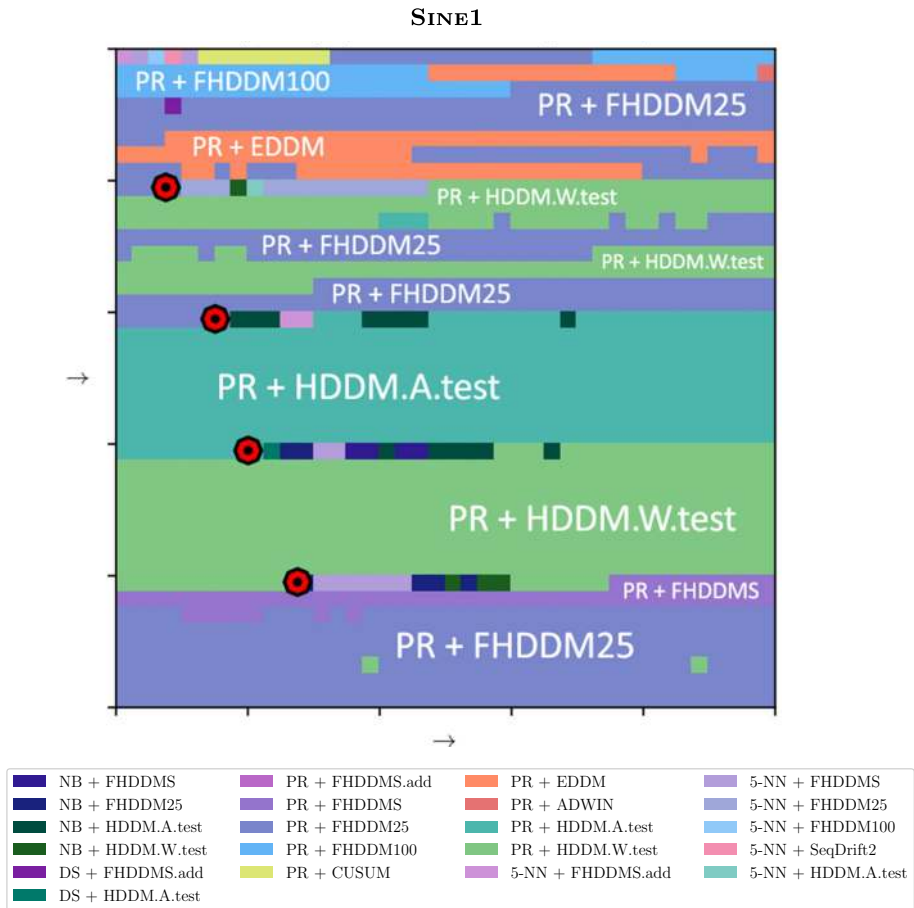


Fig. 9 Recommended Classifier+Detector Pairs out of 60 Pairs against SINE1 Data Stream with Abrupt Concept Drifts

defined in Eq. (3.7), as instances are sequentially processed over time. The experimental results as well as the recommended (classifier, detector) pairs against various data streams are presented in the following subsections.


6.5.1 Synthetic data streams

This subsection summarizes our results against the synthetic data streams, when all the (classifier, detector) pairs are executed in parallel, and are ranked over time. First, we present the top ten (10) pairs, ranked by the highest average scores, for the SINE1 data stream. Then, we illustrate the recommended (classifier, detector) pairs for all synthetic data streams in Fig. 9, B.1, and B.2. Note that the results for the other synthetic data streams are listed in the Electronic supplementary material.

Table 10 Top 10 pairs with highest average scores against SINE1

Pair	SINE1						
	Error-rate	Delay	FP	FN	Mem.	Runtime	Score
PR+FHDDM ₂₅	14.27	36.75	0	0	9.23	217.38	0.98
PR+FHDDM ₁₀₀	14.24	47.5	0	0	9.88	244.8	0.98
PR+FHDDMS	14.27	36.75	0	0	10.06	342.69	0.98
PR+HDDM _{A-test}	14.26	45.0	0	0	9.33	307.03	0.98
PR+HDDM _{W-test}	14.42	31.5	1	0	10.69	300.84	0.98
NB+FHDDM ₂₅	15.42	37.75	0	0	36.65	187.44	0.97
PR+FHDDMS _{add}	14.26	51.5	0	0	9.36	279.65	0.97
NB+HDDM _{A-test}	15.41	50.5	0	0	36.74	269.86	0.97
NB+FHDDMS	15.42	37.75	0	0	37.47	315.46	0.97
NB+FHDDMS _{add}	15.39	51.5	0	0	36.78	282.7	0.96

Top 10 (Classifier, Detector) Pairs—The pairs with the highest *average* scores against the SINE1 data stream is shown in Table 10. The table lists the results of classification error-rate, drift detection delay, drift detection false positive and false negative rates, total memory usage and runtime from left to right. Please note that the memory usage and the runtime are recorded in *kilobytes* and *milliseconds*, respectively. For this case, Perceptron (PR) and Naive Bayes (NB) paired with FHDDM₂₅, FHDDM₁₀₀, FHDDMS, HDDM_{A-test}, and HDDM_{W-test} obtained the highest average scores. This is due to the fact that they had lower error-rate, shorter detection delay, and reasonable resource usage compared to other pairs. In general, pairs with Naive Bayes had higher memory usage compared to pairs that utilized Perceptron; however, they were faster in terms of runtime.

Illustration of Pair Recommendation—The pairs recommended over time by the TORNADO framework against the synthetic data streams with *abrupt* and *gradual* concept drifts are indicated in Fig. 9 and B.1 and Fig. B.2, respectively. We utilize different colors, represented in a foursquare form of time unit, in order to show the recommended pairs over time. The pairs that use members of the FHDDM family as drift detectors are indicated in shades of *blue*, while the HDDM variants are displayed in shades of *green*. The other drift detector pairs are displayed in shades of *yellow* (for CUSUM and PageHinkley), *orange* (for DDM and EDDM), *red* (for ADWIN), and *pink* (for SeqDrift2). In our figures, the time line begins at the top-left corner and then unfolds, line by line, from left to right. The locations of the drift points in the data streams are indicated by the symbol . This symbol may thus be used to locate the drift points in the diagrams.

We set all weights to 1 for the experiments against the SINE1, SINE2, MIXED, and STAGGER data streams. Recall that, when all weights are set to 1, it is assumed that the corresponding quantities are of equal importance. The results confirm that no pair outperforms the others, and that no drift detector or classifier dominates. Initially, there are larger fluctuations in recommended pairs. In summary, Fig. 9 and B.1 show that, in general, the pairs with the FHDDM/S and HDDM variants are often recommended for having higher scores. The results also show that the Naive Bayes and Perceptron classifiers are often preferred, since they are light, fast, and accurate, particularly when the weights are equal. This is, however, not the case when the weights are distinct, for instance, as shown in Fig. B.2.

Figure B.2a illustrates the impact of the weight vector against the CIRCLES and LED data streams, that are susceptible to *gradual drift*. The pairs NB+PageHinkley, NB+FHDDM₁₀₀, and NB+HDDM_{W-test} outperform the others for the CIRCLES data stream when all the weights are set to one. In contrast, when $\vec{w} = [1.5 \ 1 \ 2 \ 1.5 \ 0 \ 0.5]^T$, the pairs of HT+HDDM_{W-test} and HT+FHDDM_{add} dominate the others for the second of half of the stream. In this case, the memory and runtime become less important, and the resulting best pair reflects this change. That is, the entries were chosen so that the pairs with lower values for the error-rate, shorter detection delay, fewer false positives and false negatives obtain higher scores. Memory consumption was not taken into account as $w_m = 0$. Finally, Fig. B.2b depicts that the pairs of PR+FHDDM₂₅, NB+FHDDMS, and NB+FHDDM₁₀₀ are recommended over time against the LED data stream; when all the weights are equal to one. Alternatively, when the weight entries are set as $\vec{w} = [3 \ 0 \ 1.5 \ 1 \ 2 \ 2]^T$, the pairs of PR+FHDDM₂₅, PR+FHDDM₁₀₀, PR+PageHinkley, and also PR+DDM are recommended. In this case, memory usage and runtime are the two preferred measures, and subsequently the Perceptron classifier outperformed the Naive Bayes one.

6.5.2 Real-world data streams

Our experimental results for the real-world data streams are reported in Tables A.6–A.8 as well as in Figs. B.3 and B.4. Recall that the pairs that use members of the FHDDM family as drift detectors are indicated in shades of *blue*, while the HDDM variants are displayed in shades of *green*. The other drift detector pairs are displayed in shades of *yellow* (for CUSUM and PageHinkley), *orange* (for DDM and EDDM), *red* (for ADWIN), and *pink* (for SeqDrift2).

Tables A.6 to A.8 list the pairs with the highest *averages* scores for the ADULT, NURSERY, SHUTTLE data streams. It is often seen that the pairs of Naive Bayes (NB), Decision Stump (DS), and Perceptron (PR) with FHDDMS, FHDDMs, HDDMs, and ADWIN are among them. Although we see a few false positives (FP) for the pairs of ADWIN, their low error-rates and short drift detection delays help them to be ranked among the top 10.

Figures B.3 and B.4 show the recommended pairs while input data are processed. The reader should notice that no single pair outperforms in all cases, and that the *best* pair changes over time. The figures also indicate that the best pairs rapidly changes at the beginning of the stream, due to lack of adequate information about their performances, while the optimal pairs remain more steady towards the ends of the streams. It is also worthwhile to illustrate that a variation of the weights impact the recommendation, as it may be noted when comparing the left and right sides of Fig. B.3 and B.4. Again, the best pair is highly dependent on the weights, especially when we vary the weights of memory and runtime versus error-rate considerations.

In summary, our results confirm that the *best* (classifier, detector) pairs vary as the stream evolves, and that the choice is highly domain dependent.

7 Discussion

We introduced the CAR measure in order to monitor the overall performance of adaptive classification models against evolving data streams. We also presented the TORNADO as a framework that simultaneously runs heterogeneous pairs of classifiers and drift detectors in parallel against data streams, while continuously recommending the best performing pair

to the user. This recommendation is based on the weights assigned to the error-rates, drift detection sensitivity, runtime and memory consumption.

In addition, we extended our earlier work and detailed FHDDMS as well as its extension FHDDMS_{add} in order to better detect abrupt concept drifts associated with shorter delay as well as to reduce the number of false negatives when a gradual drift is present. FHDDMS slides a long and a short windows, that are stacked on each other, to detect concept drifts. The longer window reduces the number of false negatives, while the shorter one detects drifts faster. In this study, we restricted ourselves to two windows, though more windows could be employed. During the evaluation of drift detection methods, we observed that HDDM_{W-test} and our FHDDMS and FHDDM algorithms are comparable, in terms of various performance measures. HDDM_{W-test} outperformed FHDDMS for faster detection against abrupt concept drifts. On the other hand, FHDDMS was better suited to detect gradual concept drifts. In either case, HDDM_{W-test} showed higher false positive rates compared to FHDDMS and FHDDM.

We conducted experiments using the TORNADO framework against synthetic and real-world data streams. Our experimental setup consisted of 60 pairs of learners and detectors, each of which were evaluated in parallel against various data streams. The experimental results clearly show, as expected, that no specific pair dominates in all cases. In the vast majority of cases, the pairs that contains the Naive Bayes or Perceptron classifiers yielded the best results, when all the weights are equal. These two algorithms provide a balance between memory usage, runtime and error-rate. This stands in contrast to the Hoeffding Tree, Decision Stump and other learners. The Hoeffding Tree algorithm generally is expensive, in terms of memory consumption as the tree grows. In addition, the runtime may increase when branching decisions become difficult. The K-NN algorithm is a lazy learner, meaning that it is greedy for both memory and runtime. These two method are therefore more suitable when runtime and memory considerations are of less importance.

Overall, as represented in Fig. 9 and B.1 as well as Fig. B.2, the pairs with HDDM_{W-test} are ranked higher than those with FHDDM for data stream containing abrupt drifts, e.g. SINE2 and STAGGER; whereas, the pairs of FHDDMS and FHDDM were recommended for the data streams with gradual concept drift; e.g. the LED data stream. It is worth mentioning that the pairs of other drift detectors ranked lower, because of their longer drift detection delays and higher false positive rates.

8 Conclusion and future work

Increasingly, there is a need for near real-time adaptive learning methods to explore dynamically evolving data streams. Such algorithms should provide decision makers with realistic, just-in-time models for short-term and mid-term decision making against today's vast streams of data. These models should not only be timely, but also be accurate and able to swiftly adapt to changes in the data. Intuitively, no adaptive learning strategy outperforms others in all settings. Similarly, the effectiveness of drift detection methods is determined by the data characteristics, the types of drifts and the rates of true positives and true negatives, among others.

Based on these observations, we created a reservoir of diverse adaptive learners and drift detection algorithms, as implemented in our TORNADO framework. In our work, we considered all (classifier, detector) pairs and then utilized them to construct models in parallel. Continuously, the current 'best' model was selected and provided to the users. Further, two

new drift detector methods, namely the FHDDMS and FHDDMS_{add} algorithms, were introduced in this paper. Our extensive experimental results confirmed that the current (classifier, detector) pairs vary over time and that they are sensitive to concept drift. Further, we showed that the two FHDDMS variants outperformed the state-of-the-art, when evaluated in terms of the holistic CAR measure.

We encountered several interesting avenues of future work. In future, we plan to also compare our TORNADO framework to existing ensembles of classifiers. The incorporation of ensembles into the framework, also needs our consideration. In our current research, we implemented our TORNADO framework on a single machine. We are now designing a hybrid environment where we utilize Cloud services together with mobile devices, such as tablets. This current research is motivated by our observation that, in many settings, such as environmental impact studies and emergency response, domain experts would require the ability to not only receive up-to-date models on their mobile devices, but also to be able to build their own models locally. In this case, we foresee that the heavy ‘bulk’ analytics would be performed in the Cloud, while the mobile devices would contain personalized, lightweight algorithms. Domain experts are often eager to include their own expertise, and thus an active learning component might prove useful. In addition, we believe that, in such a scenario, the idea of combining lightweight data analytics design with hardware-driven design (Žliobaite et al. 2015b) may further lead to more efficient algorithms.

In our current work, we in essence simplified a multi-objective optimization function through linear scalarization. We are interested in extending our work to explore whether Pareto optimization could be performed in real-time. If no particular application domain is required, the optimization could be performed directly. If the optimization is application domain dependent, the multi-objective function should be supplemented with constraints. For instance, if the memory resources are limited (mobile application) and the false positives should be avoided (such as in medical applications), such a constrained multi-objective optimization may be performed with the Karush-Kuhn-Tucker (KKT) conditions. The best way to define the multi-objective function will be an object of our future investigation.

Acknowledgements The authors acknowledge that all research conducted at the University of Ottawa was done on traditional unceded Algonquin territory. The authors further acknowledge funding by the Canadian Natural Sciences and Engineering Research Council (NSERC) as well as the Ontario Trillium Scholarship (OTS). Finally, we wish to thank the anonymous reviewers for their invaluable feedback, that led us to improve this paper considerably.

References

- Bach, S. H., & Maloof, M. A. (2008). Paired learners for concept drift. In: *Eighth IEEE international conference on data mining, 2008. ICDM'08*, pp. 23–32.
- Bache, K., & Lichman, M. (2013). UCI machine learning repository
- Baena-García, M., del Campo-Ávila, J., Fidalgo, R., Bifet, A., Gavalda, R., & Morales-Bueno, R. (2006). Early drift detection method. In: *Fourth international workshop on knowledge discovery from data streams*, Vol. 6, pp. 77–86.
- Barros, R. S., Cabral, D. R., Gonçalves, P. M., Jr., & Santos, S. G. (2017). Rddm: Reactive drift detection method. *Expert Systems with Applications*, 90, 344–355.
- Barros, R. S. M., Hidalgo, J. I. G., & de Lima Cabral, D. R. (2018). Wilcoxon rank sum test drift detector. *Neurocomputing*, 275, 1954–1963.
- Bernstein, S. (1946). The theory of probabilities
- Bifet, A., & Gavalda, R. (2007). Learning from time-changing data with adaptive windowing. In: *Proceedings of the 2007 SIAM international conference on data mining, SIAM*, pp. 443–448
- Bifet, A., & Kirkby, R. (2009). Data stream mining a practical approach

- Bifet, A., Holmes, G., Pfahringer, B., Kirkby, R., & Gavaldà, R. (2009). New ensemble methods for evolving data streams. In: *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, ACM, pp. 139–148
- Bifet, A., Holmes, G., Pfahringer, B., & Frank, E. (2010). Fast perceptron decision tree learning from evolving data streams. In: *Pacific-Asia conference on knowledge discovery and data mining*. Springer, pp. 299–310
- Blackard, J. A., & Dean, D. J. (1999). Comparative accuracies of artificial neural networks and discriminant analysis in predicting forest cover types from cartographic variables. *Computers and Electronics in Agriculture*, 24(3), 131–151.
- Catlett, J. (2002). Statlog (shuttle) data set
- Domingos, P., Hulten, G. (2000). Mining high-speed data streams. In: *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, ACM, pp. 71–80
- Freund, Y., & Schapire, R. E. (1999). Large margin classification using the perceptron algorithm. *Machine Learning*, 37(3), 277–296.
- Frías-Blanco, I., del Campo-Ávila, J., Ramos-Jiménez, G., Morales-Bueno, R., Ortiz-Díaz, A., & Caballero-Mota, Y. (2015). Online and non-parametric drift detection methods based on hoeffding's bounds. *IEEE Transactions on Knowledge and Data Engineering*, 27(3), 810–823.
- Gaber, M. M., Gomes, J. B., & Stahl, F. (2014). *Pocket data mining*. Big data on small devices series: Studies in big data.
- Gama, J., Medas, P., Castillo, G., & Rodrigues, P. (2004). Learning with drift detection. In: *Brazilian symposium on artificial intelligence*, Springer, pp. 286–295
- Gama, J., Fernandes, R., & Rocha, R. (2006). Decision trees for mining data streams. *Intelligent Data Analysis*, 10(1), 23–45.
- Gama, J., Sebastião, R., & Rodrigues, P. P. (2013). On evaluating stream learning algorithms. *Machine Learning*, 90(3), 317–346.
- Gama, J., Žliobaite, I., Bifet, A., Pechenizkiy, M., & Bouchachia, A. (2014). A survey on concept drift adaptation. *ACM Computing Surveys (CSUR)*, 46(4), 44.
- Hoeffding, W. (1963). Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301), 13–30.
- Hsu, K. W. (2017). A theoretical analysis of why hybrid ensembles work. *Computational Intelligence and Neuroscience*, 2017, 1–12.
- Huang, DTJ., Koh, Y.S., Dobbie, G., & Bifet, A. (2015). Drift detection using stream volatility. In: *Joint European conference on machine learning and knowledge discovery in databases*. Springer, pp. 417–432
- Hulten, G., Spencer, L., & Domingos, P. (2001). Mining time-changing data streams. In: *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, pp. 97–106
- Kifer, D., Ben-David, S., & Gehrke, J. (2004). Detecting change in data streams. In: *Proceedings of the thirtieth international conference on very large data bases* Vol. 30, VLDB Endowment, pp. 180–191
- Kohavi, R. (1996). Scaling up the accuracy of naive-bayes classifiers: A decision-tree hybrid. *KDD, Citeseer*, 96, 202–207.
- Krawczyk, B., Minku, L. L., Gama, J., Stefanowski, J., & Woźniak, M. (2017). Ensemble learning for data stream analysis: A survey. *Information Fusion*, 37, 132–156.
- Kubat, M., & Widmer, G. (1995). Adapting to drift in continuous domains. In: *European conference on machine learning*. Springer, pp. 307–310
- Min, J. K., Cho, S. B. (2011). Activity recognition based on wearable sensors using selection/fusion hybrid ensemble. In: *IEEE international conference on systems, man, and cybernetics (SMC)*, pp. 1319–1324
- Mitchell, T. (1997). Machine learning
- Nishida, K., Yamauchi, K. (2007). Detecting concept drift using statistical testing. In: *International conference on discovery science*. Springer, pp. 264–269
- Olorunnimbe, M. K., Viktor, H. L., & Paquet, E. (2015). Intelligent adaptive ensembles for data stream mining: a high return on investment approach. In: *International workshop on new frontiers in mining complex patterns*, Springer, pp. 61–75
- Page, E. (1954). Continuous inspection schemes. *Biometrika*, 41(1/2), 100–115.
- Pears, R., Sakthithasan, S., & Koh, Y. S. (2014). Detecting concept change in dynamic data streams. *Machine Learning*, 97(3), 259–293.
- Pesaranghader, A., & Viktor, H. L. (2016). Fast hoeffding drift detection method for evolving data streams. In: *Joint European conference on machine learning and knowledge discovery in databases*. Springer, pp. 96–111
- Pesaranghader, A., Viktor, H. L., & Paquet, E. (2016). A framework for classification in data streams using multi-strategy learning. In: *International conference on discovery science*, Springer, pp. 341–355

- Roberts, S. (2000). Control chart tests based on geometric moving averages. *Technometrics*, 42(1), 97–101.
- Ross, G. J., Adams, N. M., Tasoulis, D. K., & Hand, D. J. (2012). Exponentially weighted moving average charts for detecting concept drift. *Pattern Recognition Letters*, 33(2), 191–198.
- Sakthithasan, S., Pears, R., & Koh, Y. S. (2013). One pass concept change detection for data streams. In: *Pacific-Asia conference on knowledge discovery and data mining*. Springer, pp 461–472
- Salgado, R. M., Pereira, J. J., Ohishi, T., Ballini, R., Lima, C., & Von Zuben, F. J. (2006). A hybrid ensemble model applied to the short-term load forecasting problem. In: *International joint conference on neural networks, 2006. IJCNN'06*, pp. 2627–2634
- Sebastião, R., Gama, J., Mendonça, T. (2017). Fading histograms in detecting distribution and concept changes. *International Journal of Data Science and Analytics*, pp. 1–30
- Verikas, A., Kalsyte, Z., Bacauskiene, M., & Gelzinis, A. (2010). Hybrid and ensemble-based soft computing techniques in bankruptcy prediction: A survey. *Soft Computing*, 14(9), 995–1010.
- Vitter, J. S. (1985). Random sampling with a reservoir. *ACM Transactions on Mathematical Software (TOMS)*, 11(1), 37–57.
- Žliobaite, I. (2013). How good is the electricity benchmark for evaluating concept drift adaptation. arXiv preprint [arXiv:1301.3524](https://arxiv.org/abs/1301.3524)
- Žliobaite, I., Budka, M., & Stahl, F. (2015a). Towards cost-sensitive adaptation: When is it worth updating your predictive model? *Neurocomputing*, 150, 240–249.
- Žliobaite, I., Hollmen, J., Koskinen, L., & Teittinen, J. (2015b). Towards hardware-driven design of low-energy algorithms for data analysis. *ACM SIGMOD Record*, 43(4), 15–20.
- Žliobaite, I., Pechenizkiy, M., & Gama, J. (2016). An overview of concept drift applications. In: *Big data analysis: New algorithms for a new society*. Springer, pp. 91–114
- Zupan, B., Bohanec, M., Bratko, I., & Demsar, J. (1997). Machine learning by function decomposition. In: *ICML*, pp. 421–429